IMPLEMENTATION OF THE DATA ENCRYPTION STANDARD

A Thesis Submitted
In Partial Fulfillment of the Requirements
for the Degree of

MASTER OF TECHNOLOGY

by

N. RAVI SHANKAR

to the

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

SEPTEMBER, 1987

CERTIFICATE

This is to certify that the work entitled, "Implementation of Data Encryption Standard" has been carried out by N. Ravi Shankar under my supervision and has not been submitted elsewhere for a degree.

K.R. Srivathsan

September, 1987

(K.R. Srivathsan)
Assistant Professor
Department of Electrical Engineering
Indian Institute of Technology Kanpur

ACKNOWLEDGEMENTS

I acknowledge here with gratitude and a deep sense of appreciation, the guidance received from Dr. K.R. Srivathsan during the course of the project. He has been tolerant of many of my irregularities, often treating them with fatherly connivance, which can only inspire respect and affection.

I also wish to thank Dr. M.U. Siddiqi who initiated me to the subject of cryptography.

The list of friends who made my stay at IIT Kanpur memorable is too long, but the names of Neelu, Arvind, Raghavan and Mrs. and Mr. Srinivas cannot escape mention - some of the pleasantest times of my stay here were spent in their company.

Finally, I thank Sri U.S. Mishra for his patience with the typing of this manuscript.

ABSTRACT

A review of cryptographic systems for data communications has been carried out. The Data Encryption Standard has been implemented in hardware which interfaces with the 8085 Microprocessor workstation and in software for the IBM personal computer. Plaintext messages can be encrypted using either implementation and transmitted over a serial link to be decrypted at the other end. DES has been implemented in the block cipher mode (ECB method).

CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Data Security

Data security is the science and the study of methods of protecting data in communication and computer systems. It embodies, basically four kinds of controls [1]:

1. Cryptographic controls
2. Access controls
3. Information flow controls
4. Inference controls.

1. Cryptographic Controls:

There are two principal objectives: secrecy (or privacy), to prevent unauthorized disclosure of data, and authenticity, to prevent unauthorized modification of data. The method mainly consists in transforming (scrambling) the data using transpositions and substitutions under a suitably chosen key.

Depending on how the key is handled in the cryptosystem, we have two kinds of cryptography:

(a) Public key cryptography, where the key involved has two complementary parts, one of which is made public in a directory and the other is exclusive knowledge of the user. Encryption of a message intended for a particular user is carried out using his public key whereas decryption can be

done by the user using his private key. There is no way the user's decipherment key can be accessed with the knowledge of his public encipherment key.

(b) Secret key cryptography, where two users agree upon a key prior to the start of an information transfer, otherwise the sender communicates the key over a secure channel to the intended receiver and encrypts the plaintext with that key and transmits over the insecure channel. The Data encryption standard (DES) for instance, has been designed to suit the needs of secret key cryptography, the algorithm uses a 56 bit key to operate on a 64 bit plaintext (ciphertext) to give an output of a 64 bit ciphertext (plaintext).

2. Access Controls:

Access controls ensure that all direct accesses to stored information are authorized. By regulating the reading, changing and deletion of data and programs access controls can prevent accidental and malicious threats to secrecy and authenticity.

The effectiveness of access controls rests on two premises. The first is proper user identification. This is met through authentication procedures at login —— having password files encrypted with one way ciphers is one of the common procedures. The second premise is that the information specifying the access rights of each user or program is protected from unauthorized modification - this is usually taken care of by the operating system of the computer or database system.

3. Information Flow Controls:

Information flow controls deal with classifying information itself into security classes. An information flow policy is defined by a lattice(SC, $\leq$), where SC is a finite set of security classes, and $\leq$ is a binary relation partially ordering the classes of SC. Flow controls are concerned with the right of dissemination of information, irrespective of what object holds the information; they specify the channels or processes along which information may flow

4. Inference Controls:

They are methods intended to prevent acquisition of original information from declassified versions of confidential data. Statistical data bases, for instance provide access to statistics about groups of individuals, while access to information about any particular individual may be classified. Sometimes, by correlating statistics, original information can be gleaned; inference controls are steps taken to ensure that such attempts fail.

1.2 Scope of the Present Work

This thesis mainly addresses itself to the problem of data security over communication channels. An implementation of a Data Encryption Standard based cryptographic system has been reported here.

The DES is an algorithm originally developed by IBM of USA and thereafter prescribed by National Bureau of Standards (US) as an encryption standard for all US Governmental information transfers using the digital channel as the medium.

The algorithm itself is public knowledge and it is in fact not the secrecy of the algorithm which brings about security to the information encrypted with it, but the enormity of the computational cost involved in accessing the plaintext data from the intercepted ciphertext without access to the encipherment key.

There are many hardware packages with a varied range of options which implement the algorithm in LSI, one such device being Intel's 8294 A. The system described here consists, in part, of a hardware communication interface to the 8085 based microprocessor workstation and in its other part, it consists of software written for the IBM personal computer which implements the algorithm independently.

Besides, the thesis examines the subject of cryptography in the context of computer communication and also throws light on the many variations in which it is possible to use the DES.

1.3 Organization of the Thesis

Chapter 2 presents a survey of cryptographic systems in general and in the context of computer communication networks.

Chapter 3 discusses the DES in detail. Besides presenting the algorithm it comments on the pros and cons of using the algorithm and highlights the various modes of the use of DES.

Chapter 4 discusses the hardware communication interface to the microprocessor workstation which incorporates the Intel 8294-A IC for encryption/decryption.

Chapter 5 discusses the software implementation of the algorithm in TURBO PASCAL language and the software necessary for handling the communication between the IBM PC and the 8085 based microprocessor workstation.

Chapter 6, the concluding chapter reports the result and suggests methods to make the systems more realistic and sophisticated.

CHAPTER 2

CRYPTOGRAPHY AND SECURE COMPUTER COMMUNICATION

Advances in hardware and speed of computation have made the application of cryptographic devices to achieve the objectives of secrecy and authentication in communication and computer systems economical. In this chapter, an overview of systems which can meet the objectives of cryptography is presented. The applications of cryptographic methods to meet the security needs of digital communication networks are emphasised.

2.1 Cryptographic Systems

2.1.1 Definitions

A cryptographic system basically comprises three distinct parts: (i) an encryption device, (ii) a decryption part, and (iii) a key transfer mechanism. The flow of information in a conventional cryptographic system is shown in figure 2.1.1 The transmitter generates a "plaintext" or unenciphered message 'P' to be communicated over an insecure channel to the legitimate receiver. In order to prevent the eavesdropper from learning P the transmitter operates on P in an invertible transformation to produce the ciphertext or cryptogram $C = S_k(P)$. The key k_1 is transmitted only to the legitimate receiver via a secure channel.

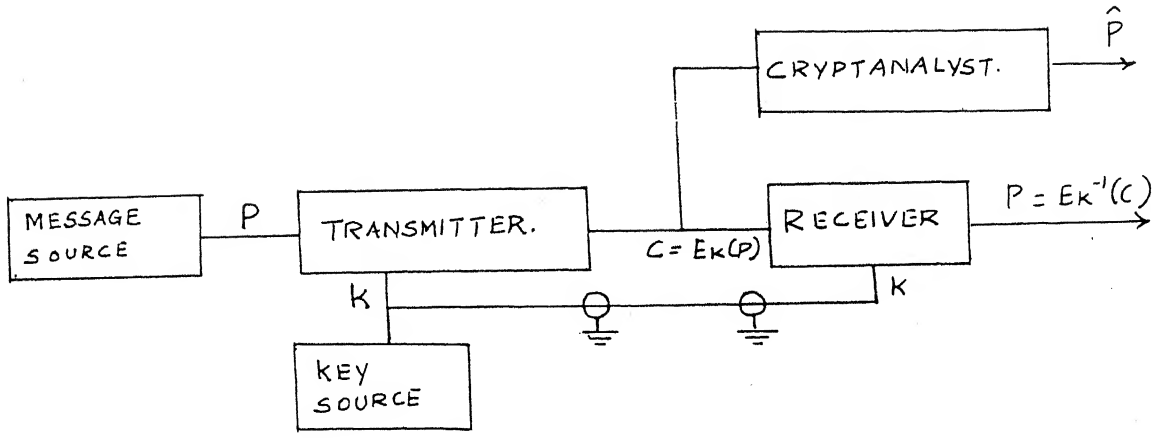


Fig.2.1.1 : Flow of Information in a Conventional Cryptographic System [3]

The goal in designing the cryptosystem $\{E_k\}$ is to make the enciphering and deciphering operations inexpensive but to ensure that any successful cryptanalytic operation (operation to deduce the message by using statistical techniques or computer aided exhaustive key search or by other methods) is too complex to be economical.

Perfect Secrecy:

A perfectly secret cipher system can be defined [2] using probabilistic notions.

A cryptographic system can be seen as a probabilistic sample space wherein:

1) Plaintext messages M occur with prior probabilities $P(M)$ where $\sum_M P(M) = 1$.

2) Ciphertext messages C occur with probabilities $P(C)$ where $\sum_C P(C) = 1$.

3) Keys k chosen with prior probabilities $P(k)$ where $\sum_K P(k) = 1$.

A necessary and sufficient condition for perfect secrecy is that for every C, $P_M(C) = P(C)$ for all M,

Where $P_M(C)$ is the probability of receiving ciphertext C given that M was sent. That is

$$P_M(C) = \sum_{E_k(M) = C} P(k) .$$

This means that the probability of receiving a particular ciphertext C given that M was sent is the same as the probability of receiving C given that any other M was sent. Figure 2.1.2 illustrates a perfect system with four messages, all equally likely, and four keys, also equally likely. Here $P_C(M) = P(M) = 1/4$ and $P_M(C) = P(C) = 1/4$ for all M and C. A cryptanalyst intercepting one of the ciphertext messages C_1, C_2, C_3 or C_4 would be at a total loss to determine which of the four keys was used.

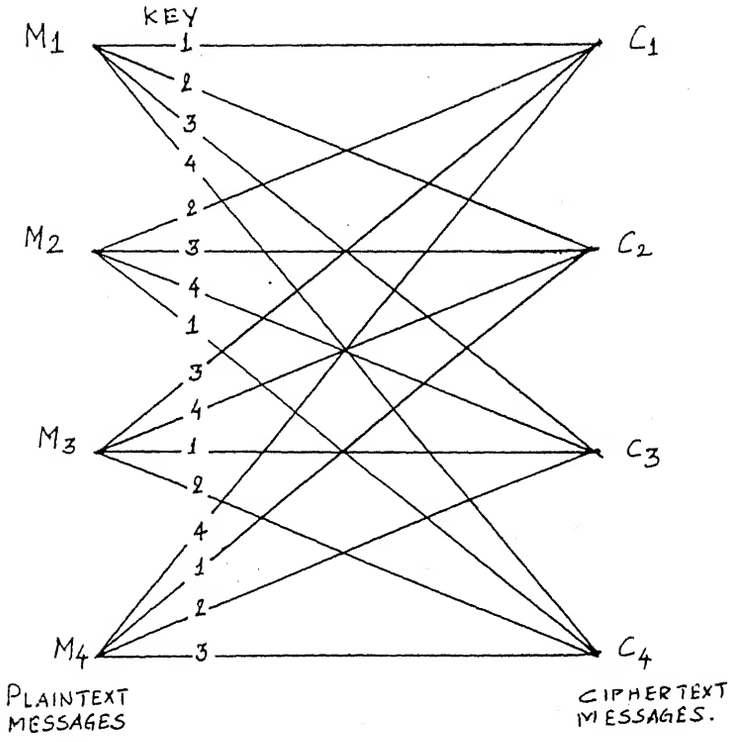


Figure 2.1.2 : Perfect Secrecy

2.1.2 Cryptanalysis [1] ,[3]

Cryptanalysis is the science and the study of methods of breaking ciphers. A cipher is breakable if it is possible to determine the plaintext or key from the ciphertext, or to determine the key from plaintext -ciphertext pairs. There are three basic methods of attack: (1) Ciphertext only (2) Known plaintext and (3) Chosen plaintext attack.

Under ciphertext only attack, a cryptanalyst must determine the key solely from the interpreted ciphertext, though the method of encryption, the plaintext language, the subject matter of the ciphertext and certain probable words may be known.

Under known plaintext attack, a cryptanalyst knows certain plaintext-ciphertext pairs. Ciphers today are considered acceptable only if they can withstand a known plaintext attack under the assumption that the cryptanalyst has an arbitrary amount of plaintext ciphertext pairs.

Under a chosen plaintext attack a cryptanalyst is able to acquire the siphertext corresponding to the selected plaintext. This is the most favourable case for the cryptanalyst.

In order to ensure the strength of a cipher, it is best to make the most adverse assumptions about the information available to the cryptanalyst. The goal must be to design computationally secure system since it is practically impossible to design an unconditionally secure system. A computationally secure system is a system which is secure due to the cost of cryptanalysis but which would succumb to an attack with unlimited computation.

The only unconditionally secure system in common use is the vernam system (also known as the one time pad) in which the plaintext is combined (ex-ored) with a randomly chosen key of the same length. But the large amount of key required makes it impractical for most applications.

2.1.3 Public Key-Vs-Conventional Cryptography

Depending upon how the key is distributed among the users of a cipher system we have two kinds of cryptographic

schemes.

1. Secret key or conventional cryptography
2. Public key cryptography.

In a conventional cryptographic system [1],[4], the key used for encipherment is exclusive knowledge of the transmitting and receiving parties. Before transmitting the enciphered message the sender ensures that the receiver is informed about the key chosen. This is done either by means of a courier or the key itself is transmitted in an enciphered form over the insecure channel.

The technique consists in effecting permutations and substitutions on the text data and producing a scrambled version that is unintelligible and undecipherable for a person with no access to the key. The ultimate secrecy of the message, of course, depends on how complex the operations carried out on the plaintext are and how secure the key is made.

The algorithm of encryption can be a standard one like the Data encryption standard or can be designed by the users in accordance with common cryptographic principles. Sethuraman [5] for instance, describes a secret key cryptographic system which operates on a 32-bit block of data. The algorithm looks upon the given block of data as an element in the vector space of 2^{32} 32 bit binary vectors and generates a permutation table which maps the given 32 bit plaintext data

into another 32 bit plaintext data. The 32 bit key describes how to arrive at a suitable permutation without having to store the entire permutation table.

Although, in principle, it is advisable for users to have their own cryptographic system owing to the fact that there may be cryptanalytic trapdoors in a public standard, yet, there is a case for using a standard such as the DES algorithm, especially in a multiuser environment, where it is impossible for each pair of users to have prior acquaintance for them to agree upon a cryptosystem. The Data Encryption Standard is a commonly used algorithm and is complex enough to defeat even sophisticated attempts at cryptanalysis. It operates on a 64 bit block of binary data and conducts sixteen rounds of permutations and substitutions on it under a 56 bit key (Described in Chapter 3 in detail).

The main feature of secret key cryptography is the generation and distribution of the secret key used for encryption/decryption. The complexity of the key-management problem sometimes proves to be a drawback in the use of secret key cryptography.

Public Key Cryptography:

Public key cryptosystems were proposed by Diffie and Hellman [3] as solution to the problem of key distribution.

A public key cryptosystem is a pair of families $\{E_k\}$, $K \varepsilon \{k\}$ and $\{D_k\}$ $K \varepsilon \{k\}$ of algorithms representing the invertible transformations:

$$E_k : \{M\} \rightarrow \{M\}$$

$$D_k : \{M\} \rightarrow \{M\}$$

on a finite message space $\{M\}$ such that (1) for every $K \varepsilon \{K\}$, E_k is the inverse of D_k. (2) for every $K \varepsilon \{K\}$ and $M \varepsilon \{M\}$, the algorithms E_k and D_k are easy to compute. (3) for almost every $K \varepsilon \{K\}$, each easily computed algorithm equivalent to D_k is computationally infeasible to compute from E_k. (4) for every $K \varepsilon \{K\}$ it is feasible to compute inverse pairs E_k and D_k from K.

Because of the third property, a user's enciphering key E_k can be made public without compromising the secret deciphering key D_k.

Given a system of this kind, the problem of key distribution is vastly simplified. Each user generates a pair of inverse transformations E and D at his terminal. The deciphering transformation D must be kept secret, but need never be communicated on any channel. The enciphering key K can be made public by placing it in a public directory along with the user's name and address. Any one can encrypt messages and send them to the user but no other than the receiver can decipher messages intended for him.

Public key cryptosystems can thus be regarded as multiple access ciphers.

An example of a public key cryptosystem which works on the above mentioned principles is the R-S-A public encrypton scheme [6], which is based on the fact that it is easy to generate two large primes and multiply them together but it is much more difficult to factor the result. Briefly, the algorithm consists in selecting two large prime numbers (100 digits long) p and q and multiplying them to produce $n = pq$. Then the Euler's function is computed as $\phi(n) = (p-1)(q-1)$. $\phi(n)$ is the number of interers between 1 and n which have no common factors with n. $\phi(n)$ as given above has the interesting property that for any integers between 0 and n-1 and any integer K

$$a^{k(\phi(n))} + 1 = a \bmod n.$$

A random number E between 3 and $\phi(n) - 1$ is then chosen which has no common factors with $\phi(n)$. This then allows $D = E^{-1} \bmod \phi(n)$ to be calculated using an extended version of Euclid's algorithm for computing the gcd of the two numbers. The information (E, n) is made public as the enciphering key and is used to transform plaintext messages into ciphertext messages as follows: a message is first represented as a sequence of integers between 0 and n-1. Then the ciphertext integer is $C = P^E \bmod n$.

The information [D,n] is used as the deciphering key to recover the plaintext from the ciphertext via $P = C^D \bmod n$. $C^D = P^{ED} = P^{K\emptyset(n)+1} = P$. As shown by Rivest et al., computing the secret, decipherment key from the public encipherment key is equivalent in difficulty to the factoring of n.

There are other methods of public key cryptography, notable among them being the trapdoor knapsack method, the algebraic coding theory method etc. [1].

2.2 Scope of Cryptographic Applications

Apart from having the obvious application affording security to transmitted messages from the threats of eavesdropping, cryptographic methods can also be used to address other problems of a related nature that arise when two parties communicate:

1. Authentication
2. The problem of disputes - digital signatures.

2.2.1 Authentication

The problem of anthentication is illustrated in the following situation.

A message passes from A to B through a communication network. We want B to know that the received message came from A and has not been changed since it left A, in other words, that it is genuinely A's message.

Also coming within the scope of authentication is the problem of authenticating the sender himself, for a message from A that has been undetectably modified is not different in its risks from an unmodified message that appeared to come from A but did not.

It is assumed that the common medium guarantees that the transmitted data is free from error using appropriate error detection and recovery procedures.

Figure 2.2.1 shows the principle of a secure authentication method. It is based on an algorithm represented by the function A(K, M). The key K is secret, and the message M to be authenticated may be of any length but the function A(K, M) must depend on every bit of the message. The function A is the authenticator that accompanies the message to its destination. The receiver also computes A(K, M) and compares this with the received value A. If they are not equal, the message is not accepted.

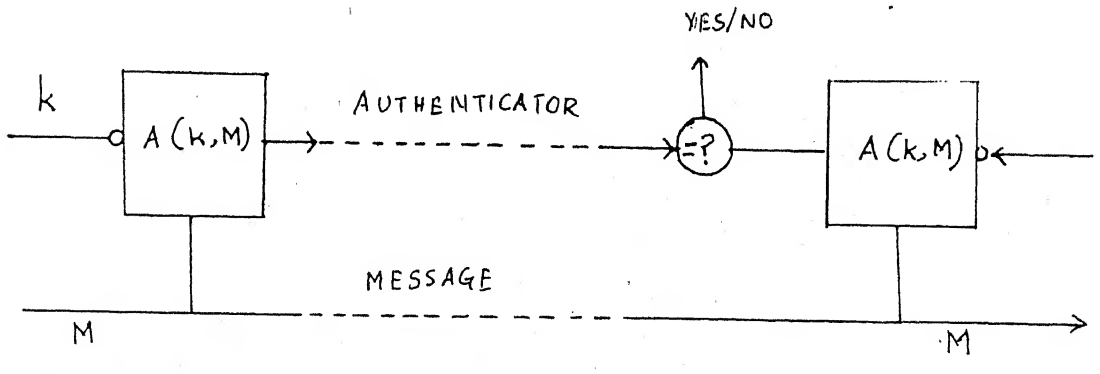


Figure 2.2.1 : The Principle of Message Authentication

Encipherment with a secret key itself provides one form of message authentication. But authentication by mere encryption cannot prevent what is known as 'spoofing' where the enemy has access to ciphertext corresponding to a fraudulent message which he can insert at a convenient place inside the cipher text. This problem can be countered by a method known as "garble extension [7] ". This means that if any portion of the cipher becomes garbled, the subsequent cipher also becomes garbled. Figure 2.2.2 shows a method by which infinite garble extension is implemented, where by spoofing prevention can be incorporated in a block encryption system.

The "E" boxes perform block encryption and the "D" boxes, block decryption. The + function indicates exclusive or. Any change to the cipher garbles the decryption of all subsequent cipher.

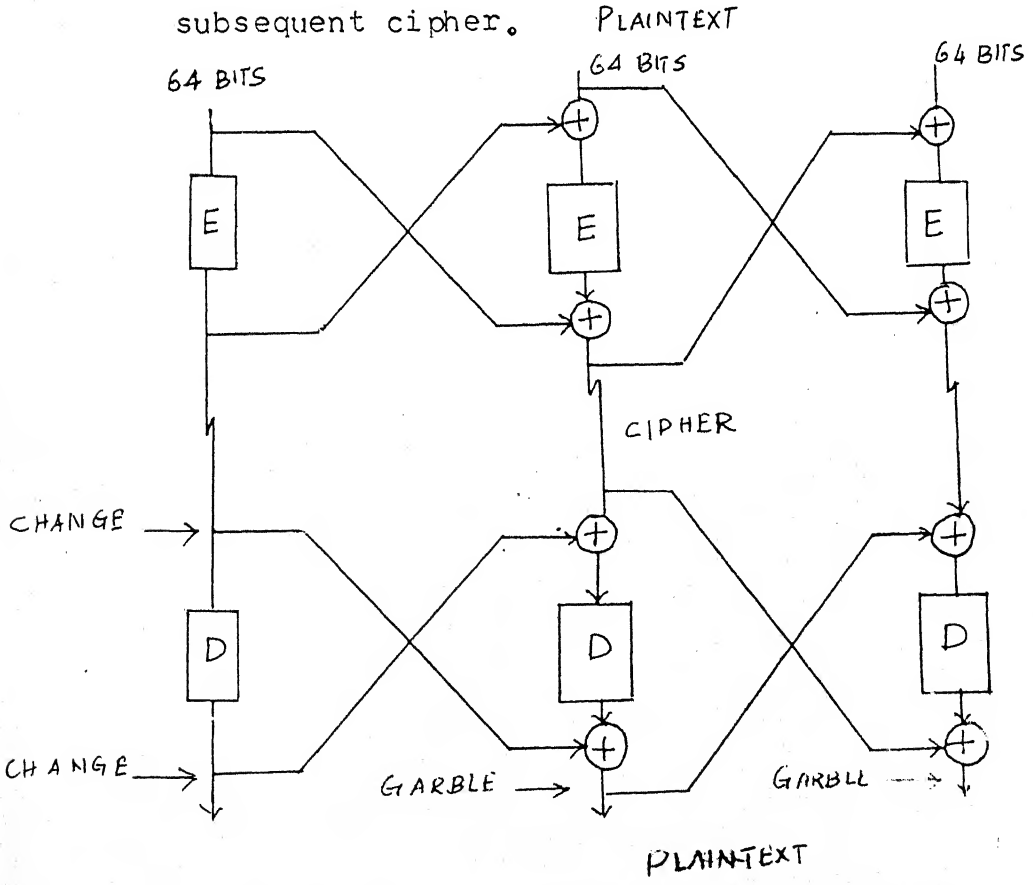


Figure 2.2.2 : Infinite Garble Extension

2.2.2 The Problem of Disputes and Digital Signatures [4]

The methods of authentication discussed above prove to be ineffective in solving the problem of disputes between the receiver and the sender over the validity of transmitted messages. Sometimes the receiver can forge messages and assert them to have been received over the channel, or the sender can maliciously deny having sent a message after having in fact done so. There must therefore a method of incorporating a digital signature in the transmitted messages which cannot be forged by the receiver and which cannot be disowned by the sender.

Public key ciphers offer an excellent solution to the problem of disputes. The sender uses his decipherment key "kd" to transform the message. This is transmitted over the channel and the receiver applies the inverse transformation 'Ke' to the received message. The sender now cannot absolve himself of having sent the message because he and he alone has access to his decipherment key. If encipherment is to be incorporated in addition to a signature the sender first 'signs' his message using his own decipherment key and then enciphers this message using the public key Ke(r) of the receiver. The receiver, as usual, deciphers the message using his own decipherment key Kd(r) and checks the signature using the public key Ke of the sender.

2.3 Computer Networks and Information Security [4],[8]

A computer network has a complex structure consisting of switching centres called nodes connected by communication links and joined to concentrators and multiplexers which provide paths to the network's host computers and terminals. To reduce their design complexity most networks are organized as a series of layers or levels, each one built upon its predecessor. The purpose of each layer is to offer certain services to the higher layers, shielding those layers from the details of how the offered services are actually implemented.

The reference model of open systems interconnection (OSI), an attempt at standardization by the OSI has seven layers. Starting at the physical layer, which carries signals from one place to another, it builds upto the top or the application layer.

Encipherment can be implemented at the bottom levels, the physical layer or the data link layer or at the top of the hierarchy in the presentation or application layers. Consequently we have two kinds of encipherment: line level encipherment, when encipherment is incorporated between nodes and end to end encipherment, when encipherment is applied only to sessions by the users.

2.3.1 Line Level Encipherment

The line is a path of communication between nodes and uses a protocol or procedure over this line. Figure 2.3.1 illustrates line level encipherment. Encryption at this level is usually implemented in hardware, using DES or some other suitable method.

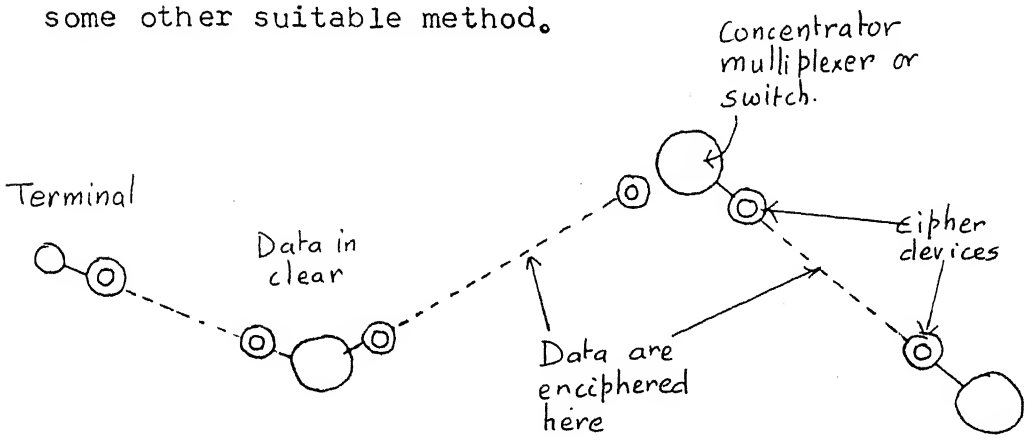


Figure 2.3.1 : Line Level Encipherment

If we encipher at this level, treating what passes over the link as a sequence of characters or bits, then an enemy who taps the line will see nothing of the structure of information. He will be unaware of the sources and destination of messages and may even be unaware of the passage of messages. This provides traffic flow security, i.e. no information about the data rate or the amount of information flowing between the nodes during a particular session is available to the enemy.

This is all the more the case if synchronous stream communication is used over a synchronous line. If, however, asynchronous start stop communication or character wide communication is used, the enemy does know the rate at which characters are passing.

The limitations of this encipherment are that data is in unenciphered form within the nodes which could be tolerated only in a private network in which the degree of care in physical protection, personnel selection, maintenance, etc. is just as high in the nodes as it is at the terminals or in the operation of the host computers. This is because while within the node, data is vulnerable to the threats of spoofing or being addressed to a wrong destination. Thus although physical and data link encryption has the advantage of providing traffic flow security, it does not provide adequate separation between one network user and another.

2.3.2 End to End Encipherment [4]

Figure 2.3.2 depicts end to end encipherment, with an encipherment device interposed between each terminal and the network with this method, the presentation or even the application layer performs the transformation using software mainly, or using special purpose hardware. End to end encryption thereby encrypts specific sessions.

When two terminals communicate with each other, a virtual communication circuit is set-up between the two terminals by the network. Consequently some of the information

present in the transmitted data is for network use and must be in unenciphered form (e.g.routing information, etc.). The 'call request' and 'call accepted' packets can be observed passing through the network, and so the progress of each call and the quantity of information carried can be monitored. There is consequently, no traffic flow security in end to end encipherment.

Thus, end to end encipherment and line encipherment provide complementary services and can be implemented together for complete security.

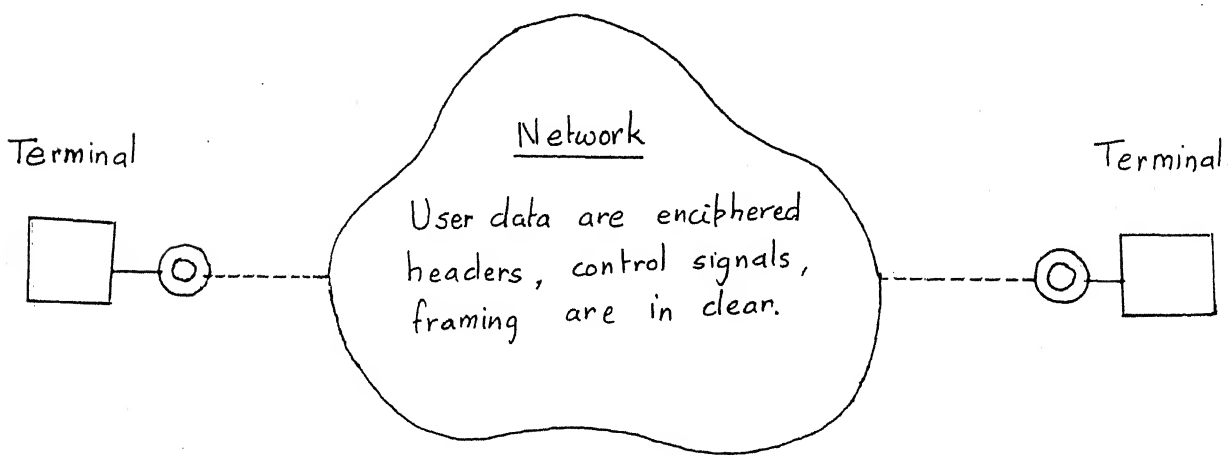


Figure 2.3.2 : End to end Encipherment

2.3.3 Key Management

Key management is one of the major problems of Network Security. It includes every aspect of handling of keys from their generation to their eventual destruction.

When encryption is incorporated at the physical or the data link level, key distribution is straight forward. The exchange of keys concerns only the two nodes connected by the line. The key can be changed at local discretion without affecting the rest of the network.

But when end to end encipherment is employed, users and terminals have to be kept separate so that they cannot interfere with one another except when they are allowed to, by the sharing of keys.

In order to move a key through a communication network it must be enciphered with another key. For example, if we have a key 'ks' which is used to encipher data and need to transport this key through the network we use another key kt to encipher it as E_{kt} (ks). The key 'ks' is typically used for enciphering data for just one session and is therefore called the session key. The key 'kt' is called the terminal key. A terminal key is used for a longer period than the session key, and it may have to be stored at a host computer with a similar number of similar keys for different terminals. To minimize the amount of secure storage needed all these can be enciphered under yet another key 'km' called the master key. Thus the storage of

'kt' is in the form of $E_{km}(kt)$.

In cases where the number of terminals is large, key storage (a total of $\frac{1}{2}$ N(N-1) keys would have to be stored for a network with N terminals) is a real problem. A network usually has a specially appointed key distribution centre (KDC) which takes on the responsibility of distributing session keys to the terminals on request.

Figure 2.3.3 shows a pair of terminals for which a session key ks is to be established. The key distribution centre must share a knowledge of the key 'kt' in common with terminal 1 and key 'kt-2' in common with terminal 2. Terminal 1 receives the enciphered form of 'ks' for its own purpose, it also receives the value of 'ks' enciphered with kt_2 which it will pass on to terminal 2 when it is establishing a call. Setting up a call has therefore two phases: obtaining the key from the key distribution centre, called the key acquisition phase and after making the call to terminal 2, transferring an enciphered form of the session key to that terminal, called the key transfer phase.

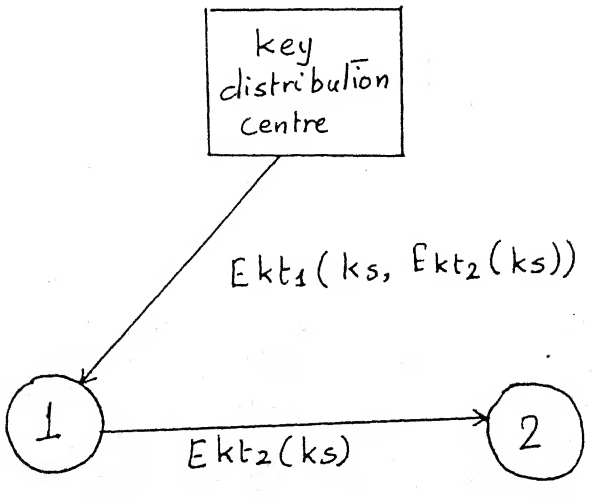


Figure 2.3.3 : Routes for Session Key Distribution

Authentication is required at both the key acqusition phase as well as the key transfer phase in order not to be affected by the consequences of replay of messages by the enemy to achieve one of the following objectives:

(1) masquerade as the k.d.c. and provide terminal 1 with a key that was distributed earlier.

(2) impersonate terminal 1 to the k.d.c. and obtain a new key for calling terminal 2, and

(3) masquerade as terminal 1 in calling terminal 2.

Authentication of the k.d.c. can be done in the following form. Terminal 1 sends to the k.d.c. in clear a_1, a_2.

Key distribution centre to terminal 1

$$E\ kt_1\ \{d/t,\ a_2,\ k_s,\ E\ kt_2\ (k_s,\ a_1,\ d/t)\}$$

a_1 and a_2 are the addresses of the two terminals, d/t stands for the current date and time.

Encryption of the current date and time under kt assures terminal 1 that the message is genuinely fresh from the k.d.c. and authentication to terminal 2 of terminal 1 is also automatically provided automatically because d/t is also encrypted using kt_2 which is transferred to terminal 2 via terminal 1.

CHAPTER 3

THE DATA ENCRYPTION STANDARD

When protected communication is required between users who belong to different organizations, some protocol regarding the encryption/decryption mechanism is inevitable. When we have a huge number of such pairs of users wishing to communicate securely, we immediately have a case for standardization of the encryption/decryption algorithm. The case is made stronger by realizing that encryption and decryption must be introduced in the context of standard communication protocols which may place new restrictions on the means of encipherment.

The DES (Data Encryption Standard) is a result of such efforts at standardization. It is a public standard, and the algorithm itself is public knowledge which means that the security provided by the algorithm is not based on the secrecy of the algorithm.

3.1 The Algorithm of DES [1]

DES belongs to a general class of ciphers called the product cipher. A product cipher E is a composition of t functions (ciphers) F_1, F_2,...,F_t, where each F_i may be a substitution or a transposition.

The operation has two 64 bit inputs, the plaintext (or ciphertext) block, the encipherment (decipherment) key block and one 64 bit output, namely, the ciphertext (or plaintext block.

A logical flow diagram of the DES is shown in Fig.3.1.1. The algorithm transforms plaintext into ciphertext or vice versa, depending on the mode in which it operates. Of the 64 bits in the encipherment key block, only 56 enter directly into the algorithm. The eight remaining bits take values for odd parity in each 8 bit byte of the key block.

As shown in Fig. 3.1.1, the input data block (64 bits long) T is first transposed under an initial permutation IP (according to table 3.1.1) giving $T_o = IP(T)$. After it has passed through 16 iterations of a function f (to be described presently), it is transposed under a final permutation IP^{-1} to give the final result (table 3.1.2).

All permutation tables should be read left to right, top to bottom. For example, IP transposes $T = t_1, t_2, \ldots, t_{64}$ into $T_o = t_{58}, t_{50} \ldots t_7$. All tables are fixed.

The key enters the algorithm in the calculation of these functions (iterations). Each of the 16 iterations uses a version of the key which is derived from its version for the previous iteration (according to key schedule calculations to be described presently). Each iteration f_i uses a 48 bit key K_i derived from K_{i-1}.

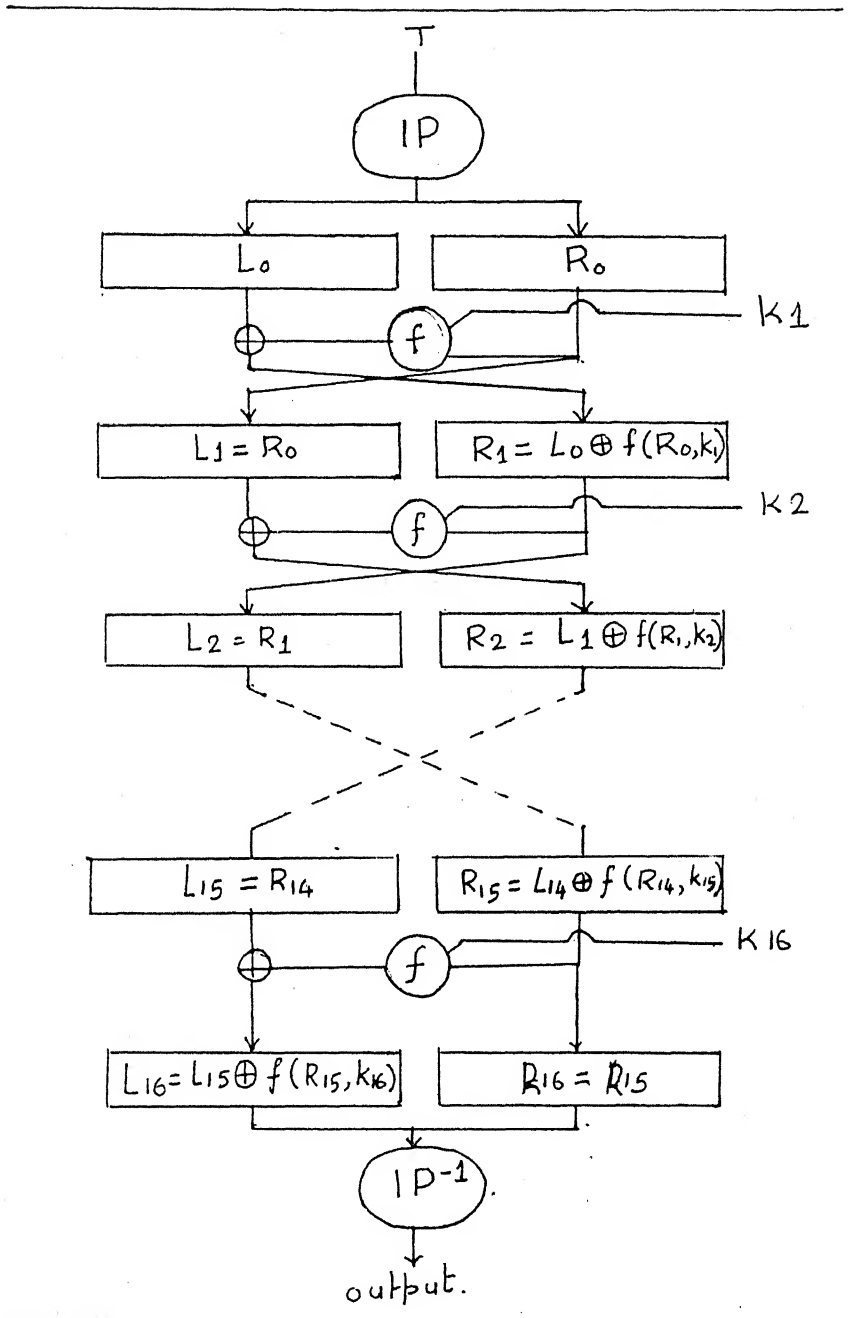


Figure 3.1.1 : Flow Diagram of the DES Algorithm

Each of the 16 iterations actually comprises

(i) The function f which combines substitution and transposition in a way that is to be described presently, and

(ii) an interchange operation 'π' which exchanges the left and the right halves of the 64 bit data.

The mechanism of the algorithm can be understood by realizing that both the function f and π are involutions [9], i.e., they are functions which are inverses of themselves. Let T_i denote the result of the i^{th} iteration and let L_i and R_i denote the left and the right halves of T_i, respectively.

That is $T_i = L_i R_i$ where

$L_i = t_1 \ldots\ldots t_{32}$

$R_i = t_{33} \ldots\ldots t_{64}$

In arriving at T_i from T_{i-1}, there is an intermediate stage, T_i' which is obtained by the transformation f_i on T_{i-1} given by

$$L_i' = L_{i-1} + f(R_{i-1}, K_i)$$

$$R_i' = R_{i-1}$$

and finally the two halves are interchanged by the operation π, where by

$$L_i = R_i'$$

and $$R_i = L_i'$$

It is easily seen that the two transformations are involutions: Enforcing the first transformation f_i on itself.

$$T_i' = L_i' R_i'$$

we get

$$T_i'' = L_i'' R_i''$$

where

$$L_i'' = L_i' + f(R_{i-1}, K_i)$$

$$= L_{i-1} + f(R_{i-1}, K_i) + f(R_{i-1}, K_i)$$

$$= L_{i-1}$$

$$R_i'' = R_{i-1}$$

which means that f_i is an involution.

Therefore between the input T and the output T_o we have

$$T_o = IP^{-1} * f_{16} * \pi * f_{15} * \pi * \ldots * \pi * f_1 * IP [T].$$

To use the same algorithm for decryption we have to supply the functions f_1 to f_{16} in the reverse order. Therefore with T_o as the input

$$T_o' = IP^{-1} * f_1 * \pi * f_2 * \pi * \ldots * \pi * f_{16} * IP [T_o]$$

substituting for T_o from the earlier expression we do see that $T_o' = T$ and we have succeeded in decrypting using the same algorithm.

58	50	42	34	26	18	10	02
60	52	44	36	28	20	12	04
62	54	46	38	30	22	14	06
64	56	48	40	32	24	16	08
57	49	41	33	25	17	09	01
59	51	43	35	27	19	11	03
61	53	45	37	29	21	13	05
63	55	47	39	31	23	15	07

Table 3.1.1
IP,the initial permutation.

40	08	48	16	56	24	64	32
39	07	47	15	55	23	63	31
38	06	46	14	54	22	62	30
37	05	45	13	53	21	61	29
36	04	44	12	52	20	60	28
35	03	43	11	51	19	59	27
34	02	42	10	50	18	58	26
33	01	41	09	49	17	57	25

Table 3.1.2
IP-1,the final inverse
permutation table.

32	01	02	03	04	05
04	05	06	07	08	09
08	09	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	01

Table 3.1.3
E,the bit selection table.

16	07	20	21
29	12	28	17
01	15	23	26
05	18	31	10
02	08	24	14
32	27	03	09
19	13	30	06
22	11	04	25

Table 3.1.4.
P,the permutation table.

```
14 04 13 01 02 15 11 08 03 10 06 12 05 09 00 07     S-1
00 15 07 04 14 02 13 01 10 06 12 11 09 05 03 08
04 01 14 08 13 06 02 11 15 12 09 07 03 10 05 00
15 12 08 02 04 09 01 07 05 11 03 14 10 00 06 13

15 01 08 14 06 11 03 04 09 07 02 13 12 00 05 10     S-2
03 13 04 07 15 02 08 14 12 00 01 10 06 09 11 05
00 14 07 11 10 04 13 01 05 08 12 06 09 03 02 15
13 08 10 01 03 15 04 02 11 06 07 12 00 05 14 09

10 00 09 14 06 03 15 05 01 13 12 07 11 04 02 08     S-3
13 07 00 09 03 04 06 10 02 08 05 14 12 11 15 01
13 06 04 09 08 15 03 00 11 01 02 12 05 10 14 07
01 10 13 00 06 09 08 07 04 15 14 03 11 05 02 12

07 13 14 03 00 06 09 10 01 02 08 05 11 12 04 15     S-4
13 08 11 05 06 15 00 03 04 07 02 12 01 10 14 09
10 06 09 00 12 11 07 13 15 01 03 14 05 02 08 04
03 15 00 06 10 01 13 08 09 04 05 11 12 07 02 14

02 12 04 01 07 10 11 06 08 05 03 15 13 00 14 09     S-5
14 11 02 12 04 07 13 01 05 00 15 10 03 09 08 06
04 02 01 11 10 13 07 08 15 09 12 05 06 03 00 14
11 08 12 07 01 14 02 13 06 15 00 09 10 14 05 03

12 01 10 15 09 02 06 08 00 13 03 04 14 07 05 11     S-6
10 15 04 02 07 12 09 05 06 01 13 14 00 11 03 08
09 14 15 05 02 08 12 03 07 00 04 10 01 13 11 06
04 03 02 12 09 05 15 10 11 14 01 07 06 00 08 13

04 11 02 14 15 00 08 13 03 12 09 07 05 10 06 01     S-7
13 00 11 07 04 09 01 10 14 03 05 12 02 15 08 06
01 04 11 13 12 03 07 14 10 15 06 08 00 05 09 02
06 11 13 08 01 04 10 07 09 05 00 15 14 02 03 12

13 02 08 04 06 15 11 01 10 09 03 14 05 00 12 07     S-8
01 15 13 08 10 03 07 04 12 05 06 11 00 14 09 02
07 11 04 01 09 12 14 02 00 06 10 13 15 03 05 08
02 01 14 07 04 10 08 13 15 12 09 00 03 05 06 11
```

Table 3.1.5.
The Substitution Boxes.

```
57 49 41 33 25 17 09
01 58 50 42 34 26 18
10 02 59 51 43 35 27
19 11 03 60 52 44 36
63 55 47 39 31 23 15
07 62 54 46 38 30 22
14 06 61 53 45 37 29
21 13 05 28 20 12 04
```

Table 3.1.6.
PC-1,Permuted choice - 1.

14	17	11	24	01	05
03	28	15	06	21	10
23	19	12	04	26	08
16	07	27	20	13	02
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Table 3.1.7.
PC-2,permuted choice - 2.

Iteration	No. of left Shifts.
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

Table 3.1.8.
Key schedule of left shifts.

Figure 3.1.2 shows a sketch of the function $f(R_{i-1}, K_i)$. First, R_{i-1} is expanded to a 48 bit block $E(R_{i-1})$, using the bit selection table E (table 3.1.3). This table is used in the same way as the permutation tables except that some bits of R_{i-1} are selected more than once, thus, given $R_{i-1} = r_1 r_2 \ldots r_{32}$, $E(R_{i-1}) = r_{32}, r_1, r_2 \ldots r_{32}, r_1$. Next; the exclusive or of $E(R_{i-1})$ and K_i is calculated and the result broken into eight 6 bit blocks $B_1, \ldots . B_2$ where $E(R_{i-1}) + K_i = B_1 B_2 \ldots . B_8$. Each 6 bit block B_i is then used as input to a selection table (substitution) function (S-BOX) S_i, which returns a 4 bit block $S_j (B_j)$. These blocks are concatenated together, and the resulting 32 bit block is transposed by the permutation P shown in the table 3.1.4. Thus the block returned by $f(R_{i-1}, K_i)$ is

$P(S_1(B_1) \ldots . S_8(B_8))$. Each S box S_j maps a 6 bit block into a 4 bit block as defined in table 3.1.5. This is done as follow S_i. The integer corresponding to $b_1 b_6$ selects a row in the table while the integer corresponding to $b_2 b_3 b_4 b_5$ selects a column. The value of $S_j(B_j)$ is then the 4 bit representation of the integer in that row or column.

Example

If $B_j = 010011$, then S_1 returns the value in row 1 and column I that is 6, which is represented as 0110.

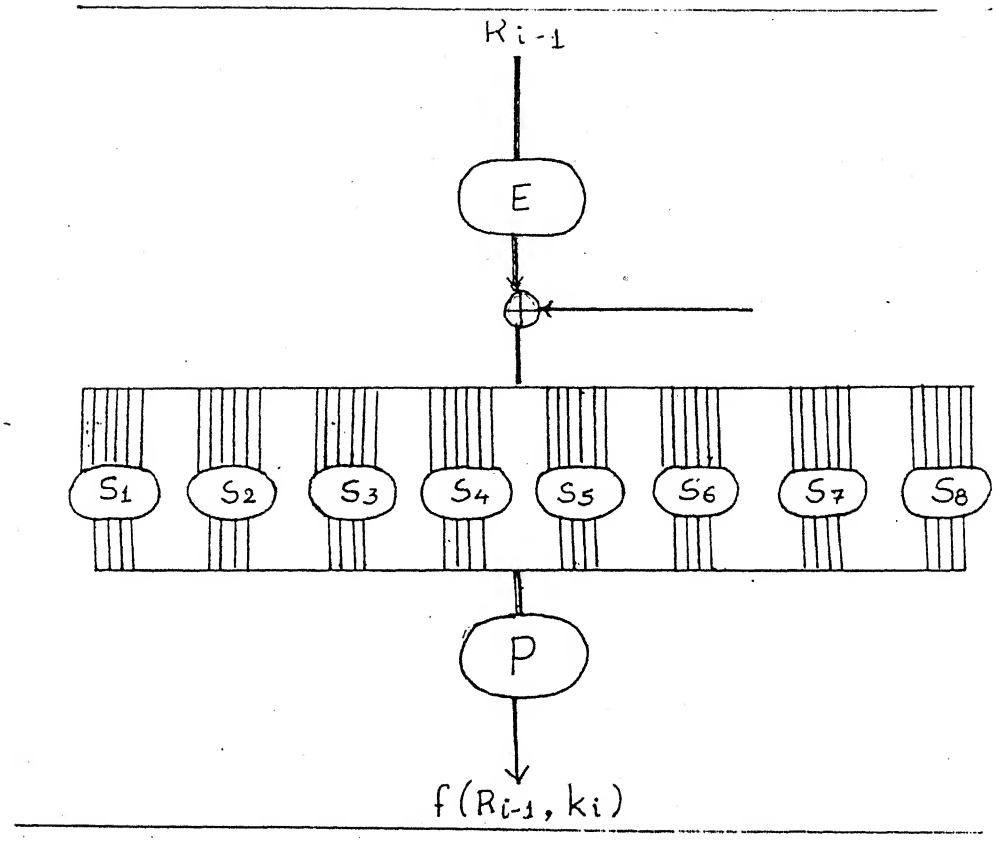


Figure 3.1.2 : Calculation of $f(R_{i-1}, K_i)$

Key calculation:

Each iteration i uses a different 48 bit key K_i derived from the initial key K. Figure 3.1.3 shows how this is done. K is input as a 64 bit block with 8 parity bits in positions 8,16,...64. The permutation PC-1 (permuted choice-1) discards the parity bits and transposes the remaining

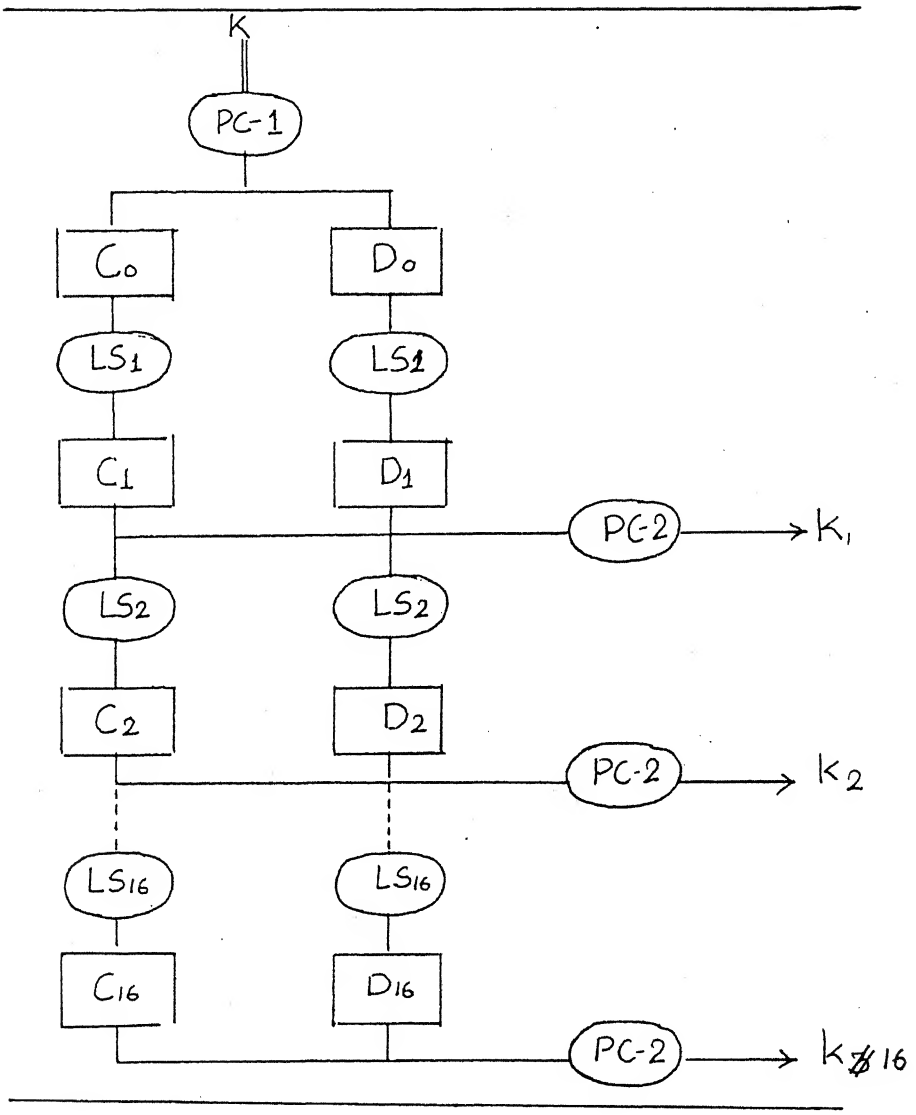


Figure 3.1.3 : Key Schedule Calculation

56 bits as shown in table 3.1.6. The result PC-1(K) is then split into two halves C and D of 28 bits each. The blocks C and D are then successively shifted left to derive each key K_i. Letting C_i and D_i denote the values of C and D used to derive K_i, we have $C_i = LS_i (C_{i-1})$ and $D_i = LS_i (D_{i-1})$ where LS_i is a circular shift by the number of positions shown in table 3.1.8 and C_o and D_o are the initial values of C and D. Key K_i is then given by K_i = PC-1 (C_iD_i) where PC-2 is the permutation shown in table 3.1.7.

3.2 Argument over the Security of DES

3.2.1 Effect of the DES Algorithm on Data

The aim of encrypting with the DES algorithm is to transform the plaintext data in such a complicated way that it is not possible to find any correlation between ciphertext and plaintext nor is it possible to show any systematic relationship between the ciphertext and the encipherment key. The effect of a change of one bit in an input plaintext block should ideally be to change the value of each individual bit in the output ciphertext block with a probability one half. DES does achieve this in practice. Please see Figure 3.2.1.

In figure 3.2.1 plaintext, ciphertext and keys are all represented in hexadecimal notation. The plaintext block is shown at the head of the table with, later in the table, other selected blocks which each differ from the first in one

key Ø 1 2 3 4 5 6 7 8 9 A B C D E F

	Plaintext	Ciphertext	Hamming distance
1	A B C D E F A B C D E F A B C D	C D E 8 7 2 D 4 A 4 7 1 3 6 4 F	
2	8 B C D E F A B C D E F A B C D	C D 3 D Ø A A 4 C 4 Ø 2 4 B 4 A	29
3	A 9 C D E F A B C D E F A B C D	8 Ø 1 F 8 A 2 9 6 8 B C 4 4 7 3	38
4	A B D D E F A B C D E F A B C D	5 D 9 8 C 4 7 D D D B A 6 F 3 Ø	36
5	A B C F E F A B C D E F A B C D	9 9 8 9 5 6 2 A 8 4 F 4 Ø 1 C 9	26
6	A B C D 6 F A B C D E F A B C D	6 7 C 2 6 9 F 2 5 4 2 7 9 1 F 9	30
7	A B C D E B A B C D E F A B C D	F 8 C 9 8 F 7 9 A D C Ø 6 E A 4	33
8	A B C D E F A B C D E F A B C D	8 7 D 3 2 4 Ø A B B F 4 4 Ø 7 4	34
9	A B C D E F A 9 C D E F A B C D	D B 9 9 8 B 6 7 Ø 4 6 C D C E 7	30
1Ø	A B C D E F A B E D E F A B C D	2 F 6 E 5 4 7 Ø E 4 E 3 5 1 A C	25
11	A B C D E F A B C C E F A B C D	B 5 3 E 4 2 D E 3 Ø F 9 7 A D Ø	29
12	A B C D E F A B C D 6 F A B C D	4 F 4 Ø 6 7 7 2 6 B 3 5 B Ø 1 4	28
13	A B C D E F A B C D E 7 A B C D	A B 1 5 5 2 8 9 6 6 Ø C 6 Ø B 2	35
14	A B C D E F A B C D E F 2 B C D	5 B D A 9 3 F 7 D 4 2 7 B 8 D 2	30
15	A B C D E F A B C D E F A F C D	9 8 5 3 C 5 1 1 E D 5 6 8 8 7 E	34
16	A B C D E F A B C D E F A B D D	7 Ø A A 2 4 Ø 7 9 5 9 F Ø 4 B 1	34
17	A B C D E F A B C D E F A B C 5	8 9 2 B E C 4 7 C 9 7 1 2 B E 3	26

Mean Hamming distance 31.Ø6

fig 3.2. DES encipherment of a series of data blocks each differing by 1 bit from a chosen block.

bit only. All these blocks have been enciphered under the same key - (0123456789ABCDEF).

A glance at the corresponding ciphertexts shows that the changes caused by 1-bit differences in plaintext are large and random. The Hamming distance is listed of each of the subsequent ciphertext blocks from the first in the table. The mean Hamming distance is 31.06 which is very close to the expected value of 32.

3.2.2 Cryptanalysis of the DES [10],[11]

The Complementation Property:

A notable feature of the DES algorithm is the property of complementation. If the complement of a plaintext block is taken and the complement of an encipherment key, then the result of a DES encipherment with these values is the complement of the original ciphertext. Thus if

$$y = E_k(\bar{x}), \text{ then } \bar{y} = E_{\bar{k}}(\bar{x}) .$$

The effect is entirely due to the presence of two ex OR operation one of which precedes the S boxes in the logical flow of the algorithm and the other which follows the permutation P.

Under a known plaintext attack the complementation property of the DES algorithm does not represent a serious weakness in the security of systems using the DES, i.e., there is no reduction in the time taken for exhaustive key search, given a known plaintext-ciphertext pair. If, however, a chosen plaintext attack can be mounted, then, by exploiting

the complementation property, the time taken to exhaust the key domain by search may be reduced by a factor of two. For instance, given the plaintext x and the values of $y_1 = E_k(x)$ and $\bar{y}_2 = E_k(\bar{x})$ so that $y_2 = E_{\bar{k}}(x)$, if all values of k are searched to find if $E_k(x)$ equals y_1 or y_2 then each test covers the two key values K and $\bar{K}$.

Steps must therefore be taken to ensure not to enable an opponent to discover both $E_k(x)$ and $E_k(\bar{x})$.

Exhaustive Search for a DES Key:

Proceeding on the assumption that the cryptanalyst has some how obtained matching blocks of plaintext and ciphertext, the next step is to test all possible keys by enciphering the plaintext in turn with each and comparing the result with the known, ciphertext. When a match is obtained between the produced and the known ciphertext the current key value is that being sought. The time taken to exhaust the whole of the key domain will depend on the time taken to carry out the DES encipherment. If we assume that the time taken for the encipherment is 100 ms and that only one device is used, then we can calculate the time required to test all possible keys to be 7.2×10^5 S or about 228 million years. If we assume that the encipherment device is some what faster, with an operation time of 5 μs, then the total time required to exhaust the key domain is just over 11000 years. Clearly these times are totally impracticable for a cryptanalyst.

The exhaustive search times only become practicable when we have key searching which uses many DES devices in parallel, each of which is searching a different part of the key domain. It has been estimated that the time can be reduced to as low as 20 hours with a machine costing $ 72 million which uses 1 million DES devices acting parallelly each requiring 1 μs to do an encryption operation. Such search strategies, however, can still be frustrated with techniques such as cipherblock chaining (next section).

All these techniques, of course, are of the brute force kind, i.e., assuming that the cryptanalyst does not have access to any trapdoor information that might possibly have been introduced into the system during the design of the algorithm. But there are speculations that the actual DES-S-boxes contain some deliberate weaknesses put there in order to make the algorithm subject to a 'trapdoor' attack by those in possession of the design information.

3.3 Modes of Block and Stream Encryption Using the DES [4]

Block encipherment operates on blocks of data of fixed size but a message to be enciphered can be of any size. A block cipher breaks a plaintext message M into successive blocks M_1, M_2 and enciphers each M_i with the same key K, i.e.,

$$E_k(M) = E_k(M_1)\, E_k(M_2) \ .$$

A stream cipher breaks the message M into successive characters or bits m_1, m_2,.... and enciphers each with the i^{th} element K_i of a key stream $K = K_1, K_2, \ldots$ that is

$$E_k(M) = E_{k_1}(M_1)\ E_{k_2}(M_2) \ .$$

DES can be used for encryption either in the block mode or in the stream mode. There are four standard ways of implementing an encryption/decryption system using DES:

(1) Electronic Codebook Mode

(2) Cipher Block Chaining

(3) Cipher Feedback

(4) Output Feedback

3.3.1 The Electronic Code Book (ECB) Mode:

The ECB method of encryption uses DES in the 'native' block cipher mode, dividing the entire message into 64 bit blocks, and enciphering each one separately. It is generally considered the weakest form of encipherment because of the fact that it does not connect the blocks together owing to which repeated phrases which happen at the same phase relative to the block size will show through in the ciphertext. Cryptanalysts will search for and exploit these occasional regularities.

3.3.2 Cipher Block Chaining (CBC):

Cipher block chaining uses the output of one encipherment step to modify the input of the next, so that each cipher block is dependent, not only on the plaintext block from which it immediately came but also on all previous plaintext blocks. Figure 3.3.1 illustrates how it operates.

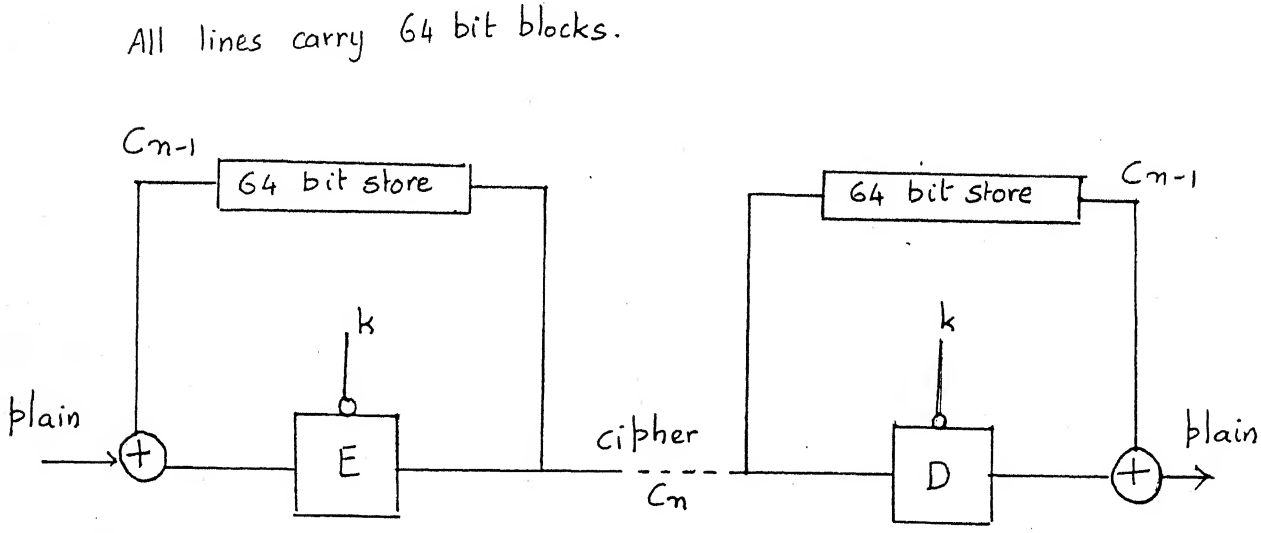


Figure 3.3.1 : Cipher Block Chaining

The operation of cipher block chaining can be expressed as follows:

Encipherment: $C_n = E_k (P_n + C_{n-1})$

Decipherment: $Q_n = D_k (C_n) + C_{n-1}$

where P_i is the i^{th} plaintext block

C_i is the i^{th} ciphertext block

$+$ stands for mod 2 addition.

The encipherment process is started by initially loading the 64 bit storage suffer in fig. 3.3.1 with a previously agreed upon string of data known as the initializing variable. Therefore for n = 1, $C_1 = E_k (P_1 + I)$

$$Q_1 = D_k(C_1) + I$$

The iniliatizing variable I is kept secret, and is usually transmitted by encryption in the ECB mode.

Although the ciphertext corresponding to a plaintext block depends on all previous ciphertext blocks, a line error occurring somewhere does not affect all the subsequent blocks. Ths is clear on examining figure 3.1.1. The system recovers just after two successive erroneous blocks. But the user must be on his guard against synchronization errors for if a bit is lost or gained in transmission so that blocks are shifted one bit out of position, then the receiving system will generate garbage indefinitely.

3.3.3 Cipher Feedback (CFB):

Data are handled in many forms, as complete messages, or sequences of frames blocks, 8 bit characters or binary digits. When messages must be treated character by character another kind of chained encipherment is used which is known as cipher feedback. We could, however group bits into larger blocks and use one of the two methods described earlier but if we are operating at a low level in the hierarchy of computer protocols where transparency is important, we

must introduce encipherment in a way which disturbs the existing system as little as possible.

The cipher feedback method is illustrated in figure 3.3.2.

Whereas CBC operates on whole blocks, CFB operates on one character at a time, and the character length m can be chosen as a parameter of the design. It is known as m-bit cipher feedback.

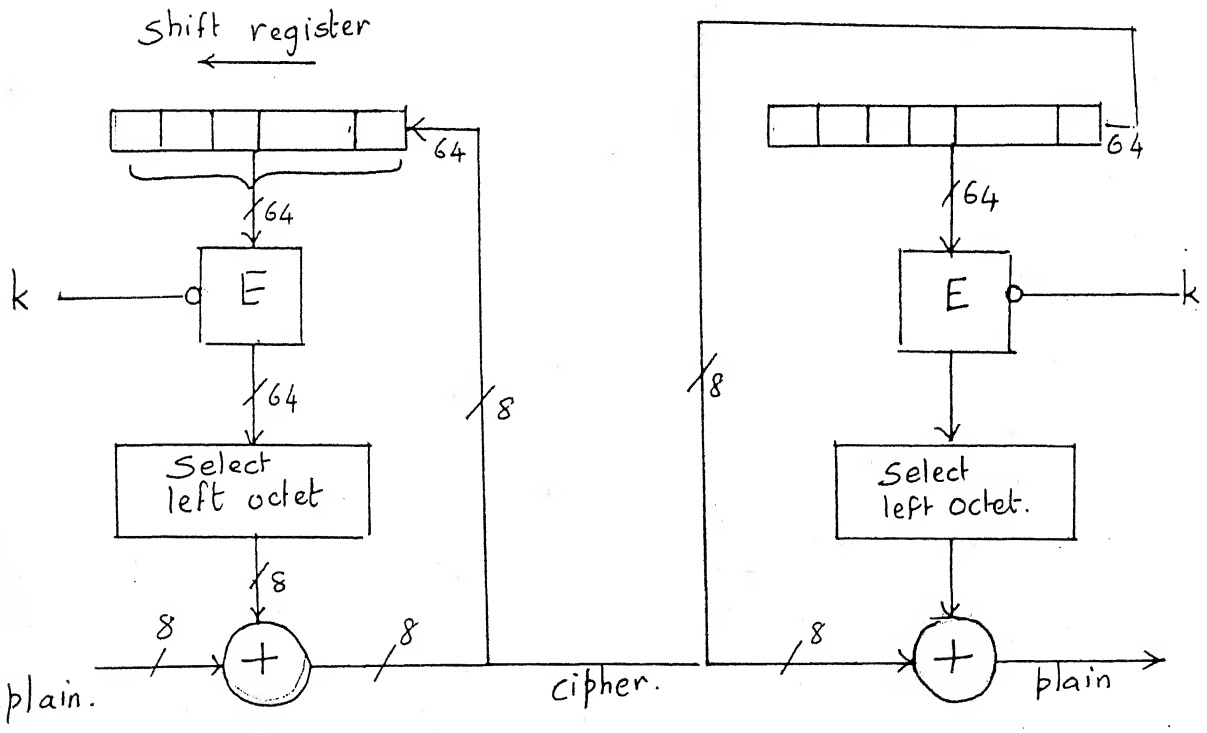


Figure 3.3.2 : Cipher Feedback

For m = 1 we have a stream cipher which would come under the broad category of self synchronous stream ciphers.

In the figure above the eight bits used to encipher the character stream by modulo 2 addition are derived from bits 57 to 64 of the output of the DES algorithm. The DES algorithm performs 'ENCIPHERMENT' at both the ends of the line. The input to the DES device comes from a 64 bit shift register which contains the most secret bits transmitted as ciphertext. Every character which goes out on the line at the sending end is shifted into the high numbered bits of the register displacing similar number of bits from the other end of the line.

Because of the chaining involved between successive characters as in cipher block chaining, line errors do propagate even with cipher feedback and in this case propagation is longer than in CBC.

Whenever the ciphertext corresponding to a character is garbled or altered due to line errors, 9 characters including the present one are erroneously deciphered (in the case of 8 bit CFB).

As in CBC, cipher feedback also does have to be initialized with an initializing variable. The IV is transmitted over the line using the ECB method.

3.3.4 Output Feedback (OFB):

The fourth method, output feedback is intended for applications in which the error extension properties of CBC and CFB are troublesome. It is a stream cipher and resembles the vernam cipher. Only here, instead of using a pseudorandom shift register generator for generating the key stream. We use a nonlinear key generator which incorporates a DES device. The method is illustrated in figure 3.3.3.

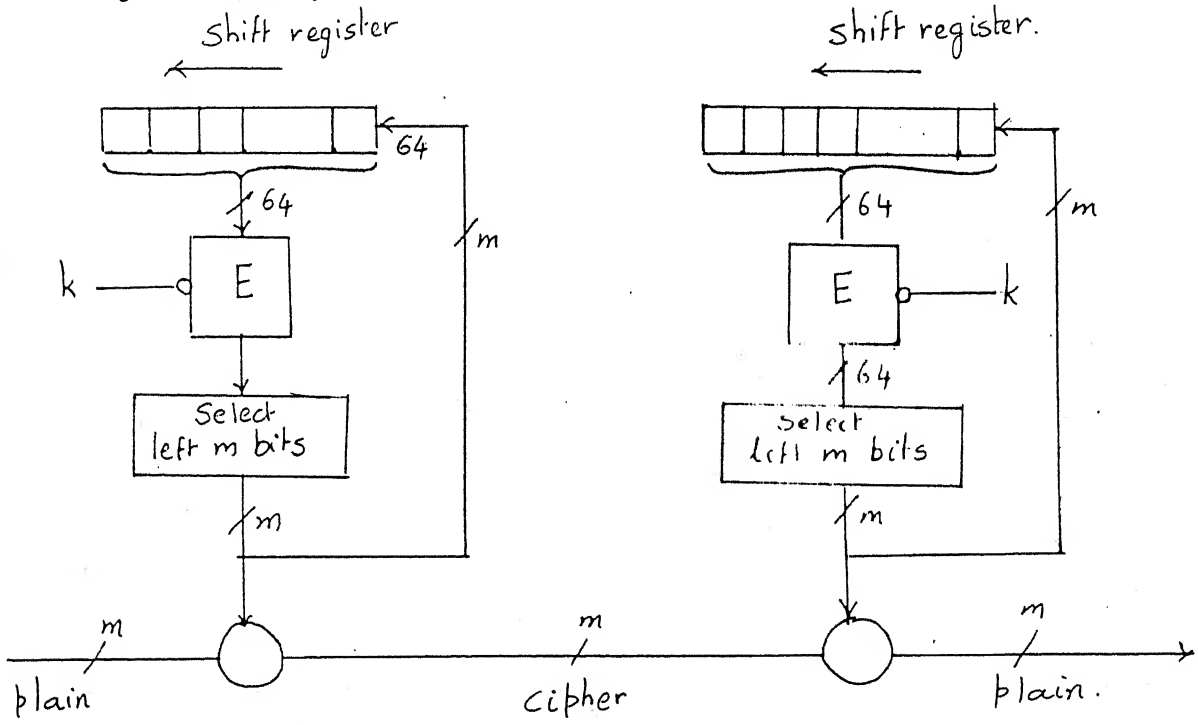


Figure 3.3.3 : m-bit output feedback.

Synchronization of the pseudo-random number generators at the two ends is more important in this case than in the CFB and CBC cases because unlike in these two cases the OFB operation will not recover if characters have been gained or lost. A system employing OFB must have a method of resynchronization after such a failure. This is the same as restarting with a new IV.

3.3.5 Summary:

Comparing the properties of the four 'standard' methods of operation using the DES we can see that the electronic code book method (the simple use of the block cipher), is not suited for messages larger than one block because of the danger of 'code book' analysis of repeated block values and the possibility of reassembling messages from known blocks. The most usual application of the ECB method is to encipher a key for transmission. Cipher block chaining is best for messages more than one block. This method avoids codebook analysis generally but not at the start of the chain. But by varying the IV reasonably frequently this problem can be circumvented. Cipher feedback is used for enciphering streams of characters when characters must be treated individually. Both CBC and CFB methods recover from errors which is to say that error propagation is finite. Output feedback is needed when error extension is undesirable.

The four methods of operation are versatile enough for nearly all applications. Still they do not exhaust the possibilities. We can construct other methods for using the block cipher on chains of blocks.

3.4 Implementation of the DES [4]:

DES can be implemented both in software and hardware. Hardware implementations achieve encryption rates of several million bps (bits/sec).

3.4.1 Hardware:

Single LSI chips embodying the DES algorithm are available from several manufacturers, notable among these being:

(i) the MC 884 (from Burroughs of Detroit, Michigan, USA) (This also requires the MC 883 for control of its operation) which is TTL compatible. The algorithm takes between 25.6 μs and 64 μs for encryption or decryption. Including the overheads required for input/output the effective data rate is in the range of 83 K byte/sec to 125 K byte/sec. The chip supports CFB operation.

(ii) The WD 2001 E/F, WD 2002 A/B and WD 2003 (all from Western Digital, Newport beach, California, USA). All inputs are TTL compatible and can be used with 8080 family of processors. For a maximum clock rate of 2 MHz the WD 2001 operates 167 K bytes/sec.

(iii) The MC 6859 (from Motorola) with a clock rate of 2 MHz, we get a data rate of 400 K bit/sec.

(iv) AMZ 8068 (from AMD of Sunnyvale, California), a powerful chip with a claimed throughput of 1.7 M bytes/sec with an algorithm time of less than 5 μs. The device interfaces with the AMZ8000 CPU bus and may also be used with the 8085 and 8084 families of processors.

(v) INTEL 8294, which aims at low cost rather than high speed, is a microprocessor peripheral device which operates only in the ECB mode giving an effective data rate of 80 bytes/sec. The chip provides for DMA make of operation. Used as a CPU peripheral no further control unit is needed for the 8294; used separately, it must be controlled by a separate microprocessor.

This thesis reports the development of a communication interface for the 8085 based Microprocessor work station using the 8294 chip for Decryption and Encryption. The details are described in the next chapter.

Software:

It becomes necessary sometimes to implement the DES algorithm in software, although computers are not at their best while doing the bit manipulations required by the algorithm; especially if encryption is to be implemented at a higher level in the hierarchy of Network architecture.

It would be ideal to write software using the assembly language of the main processor of the computer system concerned because of the obvious advantages in speed of encryption.

This thesis, however, reports the implementation in PASCAL of the algorithm for the IBM personal computer. PASCAL was chosen as the language in view of the difficulties

arising due to the complexity of the algorithm of DES and of course, the ease in coding in a high level language like PASCAL.

CHAPTER 4

THE HARDWARE COMMUNICATION INTERFACE

This chapter describes an implementation in hardware, of an encryption scheme which incorporates the INTEL-8294 A IC for the purpose of encryption/decryption. The card interfaces directly with the 8085 based microprocessor work station; it uses the CPU of the work station for control of the main encryption device as well as the associated circuitry for serial communication with another terminal (Here, the IBM PC). The assembly language program for handling the data transfers to and from the interface card resides in the workstation RAM and must be loaded into it every time the card is to be used, owing to the absence of an EPROM on the card. The data (message) to be encrypted/decrypted also resides in the workstation memory. The workstation, therefore, allowing the limitations on the size of its memory, can be used as an independent terminal for transmitting encrypted messages to a distant terminal via the interface card.

4.1 The Encryption Device 8294-A [14]

The Intel 8294-A Data Encryption Unit (DEU) is a programmable microprocessor peripheral device designed to encrypt and decrypt 64 bit blocks of data using the DES algorithm. The DEU operates on 64 bit textwords using a 56 bit user specified key to produce 64 bit cipher words.

The operation is reversible, that is, the chip can be programmed to operate in the 'Decrypt' mode as well whereby it operates on the cipher word to produce the original textword. The mode of operation can be changed at any stage by using a single byte command.

The algorithm itself is permanently contained in the 8294-A; however, the 56 bit key is user defined and can be changed at any time. Figure 4.1.1 shows a block diagram depicting the internal structure of the DEU.

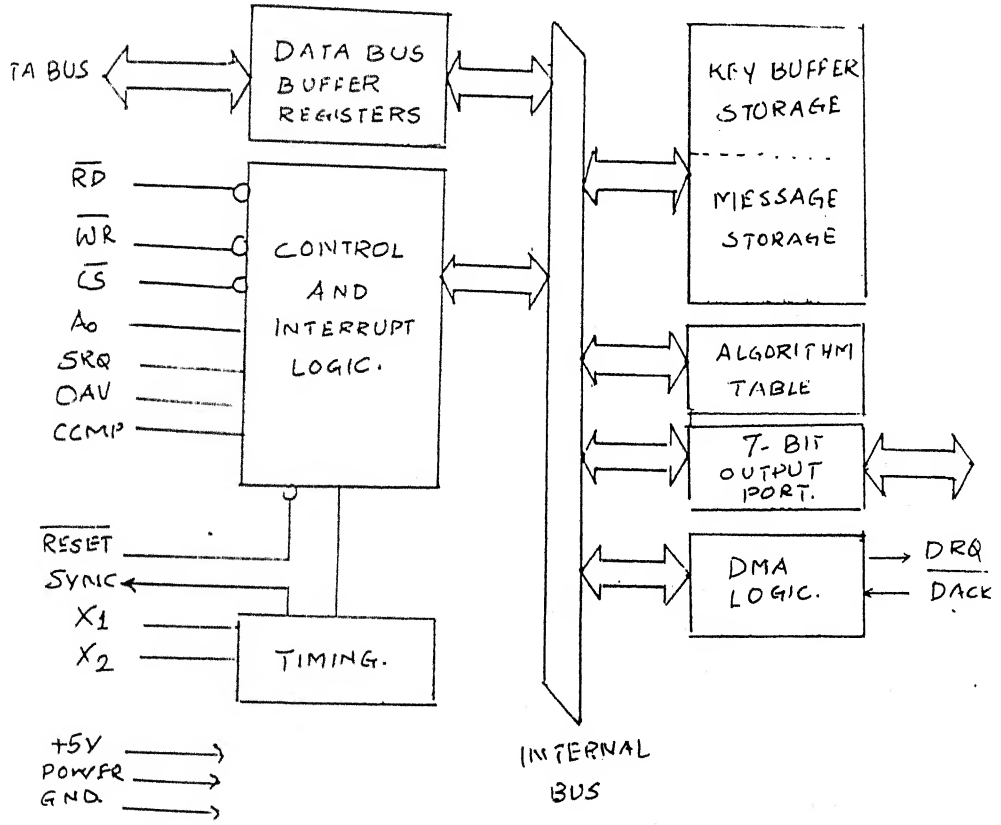


Figure 4.1.1 : Block Diagram of the 8294-A

The 56 bit key and the sixty four bit message data are transferred, to and from the 8294-A in bytes by way of the system data bus.

The chip also provides a DMA facility by means of which data can be transferred directly to the DEU without intervention of the CPU. For this purpose, two pins are provided on the IC, a DMA request output pin , and a DMA acknowledge input pin . The chip will then have to be used in conjunction with the 8257 DMA controller or any equivalent device. The circuit details are given in the next section.

Since the microprocessor workstation has a RAM of limited size, use of the DMA option for the interface card constitutes no specific advantage. The chip, however, allows to be operated in the 'Encrypt-decrypt mode', whereby it can be addressed as an input-output port. The card uses this mode of operation for data transfers to and from the DEU.

Three interrupt outputs are available to ease the load on the CPU if it is otherwise busy; they also help to minimize the software overhead associated with the Data transfer: (i) The SRQ (Service Request) interrupt which, if enabled, interrupts the CPU indicating that the 8294-A is awaiting data or commands at the input buffer (ii) the OAV (output available) interrupt, which indicates to the CPU

that the 8294-A has data or status available in its output buffer, (iii) the CCMP (conversion complete) interrupt pin, which is an interrupt to the CPU indicating that the encryption/decryption of an 8 byte block is complete.

Operation:

The data conversion sequence is as follows:

1) A set mode command is given, enabling the desired interrupts.

2) An enter new key command is issued, followed by 8 data inputs which are retained by the DEU for encryption/decryption. Each byte must have odd parity.

3) An encrypt data or decrypt data command sets the DEU in the desired mode.

After this data conversions are made by writing 8 data bytes and then leading back 8 converted data bytes. Any of the above commands may be issued between data conversion to change the basic mode of operation of the DEU, e.g. a Decrypt data command could be issued to change the DEU from encrypt mode to decrypt mode without changing either the key or the interrupt modes enabled.

Internal DEU Registers:

Four internal DEU registers are addressable by the master processor: 2 for input, 2 for output. The function of each of these registers is described below.

Data Input Buffer:

Data written into the register is interpreted in one of the following three ways depending on the preceding command sequence:

(i) part of a key (if the preceding command is an 'Enter New Key' command).

(ii) Data to be encrypted and decrypted (if the preceding command is an encrypt-data or decrypt data command).

(iii) A DMA block count (if the preceding command is a set mode command programming the DEU in the DMA mode).

Data Output Buffer:

Data read from this register is the output of the encryption/decryption operation.

Command Input Buffer:

Commands to the DEU are written into this register.

Status Output Buffer:

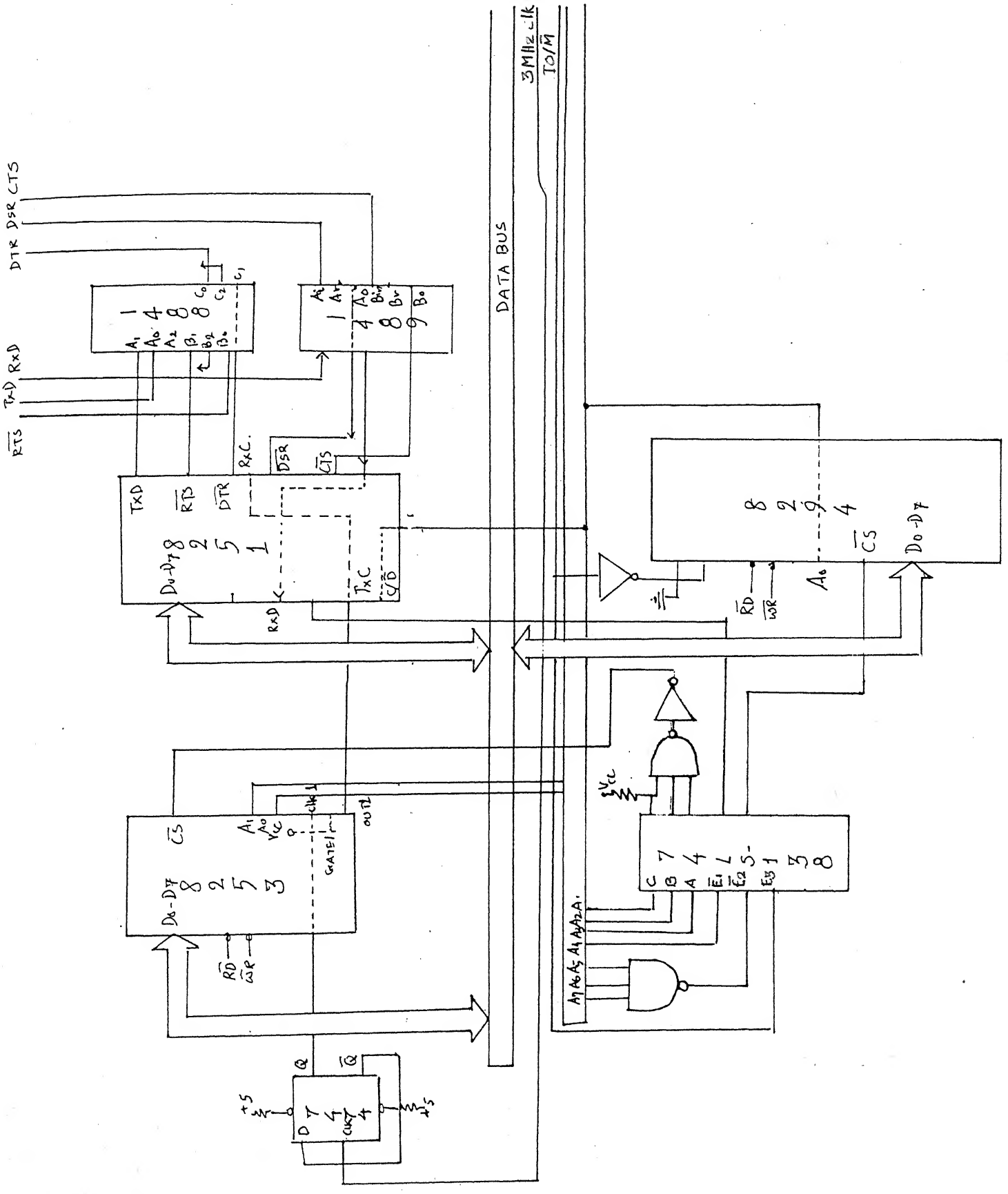
DEU status is available in this register at all times. It is used by the processor for poll driven data transfer operations. The CPU polls the flags CF (completion flag), OBF (output buffer full) and IBF (Input buffer full) from the status register and issues read write commands to the DEU accordingly.

4.2 Circuit Details:

Figure 4.2 shows the circuit diagram of the interface card. Besides the main encryption unit 8294-A, there is also a USART (Universal synchronous - Asynchronous Receiver-Transmitter - the Intel 8251-A) for serial communication between the workstation and an external computer . The clock inputs of both the 8294-A and the 8251 are connected directly to the system clock. However the transmit and the receive clock signals are obtained from the programmable timer I-8253. The timer is programmed to operate in the rate generator mode (mode 3). The clock input of the timer is derived from the system clock not directly but via 7474 D-flip flop which is configured to give a divide by 2 operation.

The address decoding logic comprises a 74LS138 3 to 8 decoder and a triple. 3-input NAND Gate (741Ø) and 7404 Hex inverter.

At the interface between the serial communication USART and the external host, there is a 1488 line driver and a 1489 line receiver which convert TTL signal levels to RS-232-C levels and vice versa respectively. These are essential because communication adapters provided with most computers adopt this standard (RS-232C) for signal levels.



4.2 : CIRCUIT DIAGRAM.

DMA Mode of Operation:

When messages spanning several hundreds of bytes are required to be encrypted using the DEU, the DMA mode of operation is very useful. The circuit diagram for data conversions using DMA is shown in figure 4.2.3.

The use of the DMA feature requires 3 external AND gates and 2 DMA channels (one for input, one for output). Since the DEU has only one DMA request pin, the SRQ and the OAV outputs are used in conjunction with two of the AND gates to create separate DMA request outputs for the 2 DMA channels. The third AND gate combines the two active low DACK inputs. To initiate a DMA transfer, the CPU must first initialize two DMA channels. It must then issue a set mode command to the DEU enabling the OAV, the SRQ and DMA outputs. Following the Set Mode command there must be a data byte giving the number of 8 byte blocks of data ($n < 256$) to be converted. The DEU then generates the required number of DMA requests to the 2 DMA channels with no further CPU intervention. When the requested number of blocks have been converted, the DEU will set the CF and assert the CCMP interrupt (if enabled). CCMP then goes false again with the next write to the DEU. Upon completion of the conversion the DMA mode is disabled and the DEU returns to the encrypt/decrypt mode. The enabled interrupt outputs, however, will remain enabled until another set mode command is

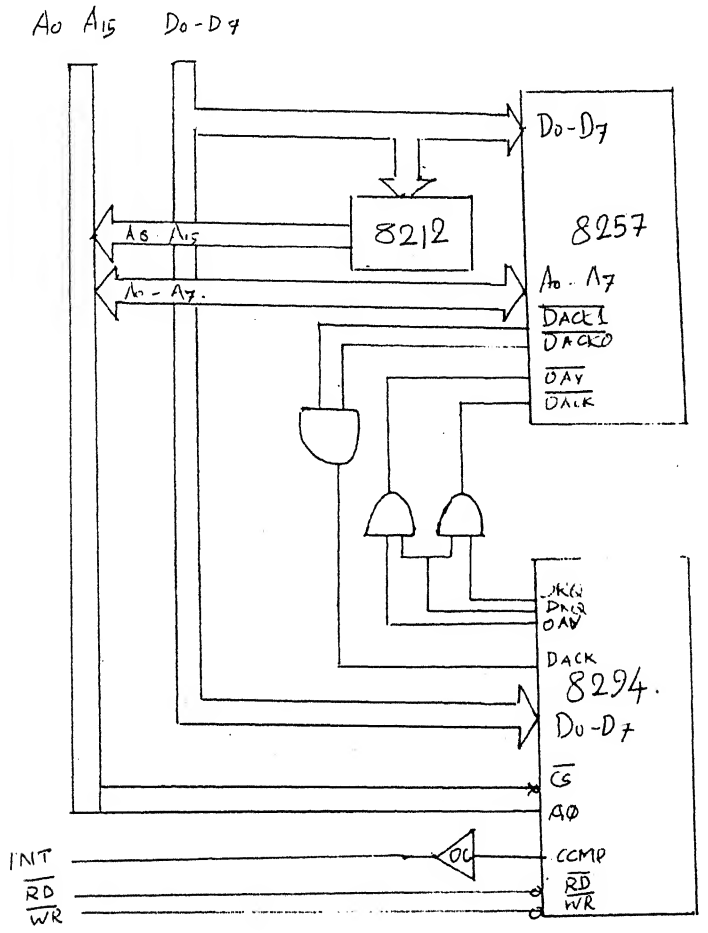


Figure 4.2.3 : DMA Interface

4.3 The Assembly Language Program:

A flow chart of the Assembly Language Program to effect Data Encryption and Decryption and to transmit/receive encrypted data over a serial link is shown in figure 4.3.1.

The program stores messages meant for encryption and decryption in different areas in memory. Registers DE and HL contain the addresses of converted data and plaintext/ciphertext data respectively.

The key is stored at fixed location in the memory. It is entered interactively from the video terminal on being prompted. The program adjusts the key for parity bits and sends it to the DEU after entering the appropriate command.

Messages meant for encryption are entered interactively and are stored at fixed locations. The program interprets every character of the message as an 8-bit byte, i.e., it sends the ASCII 7 bit code for encryption. Encryption is thus effected on 8 characters at a time.

The 8294-A is a low cost chip; it does not offer facilities of CFB or CBC modes of DES operation (Chapter 3). The program therefore encrypts data in the ECB (Electronic Codebook) method.

The program pads up the message with blank characters to make the number of characters an exact multiple of 8.

After the Data conversion is over, the encrypted data is sent over the serial link to the PC. The PC then decrypts the message in software (discussed in Chapter 5).

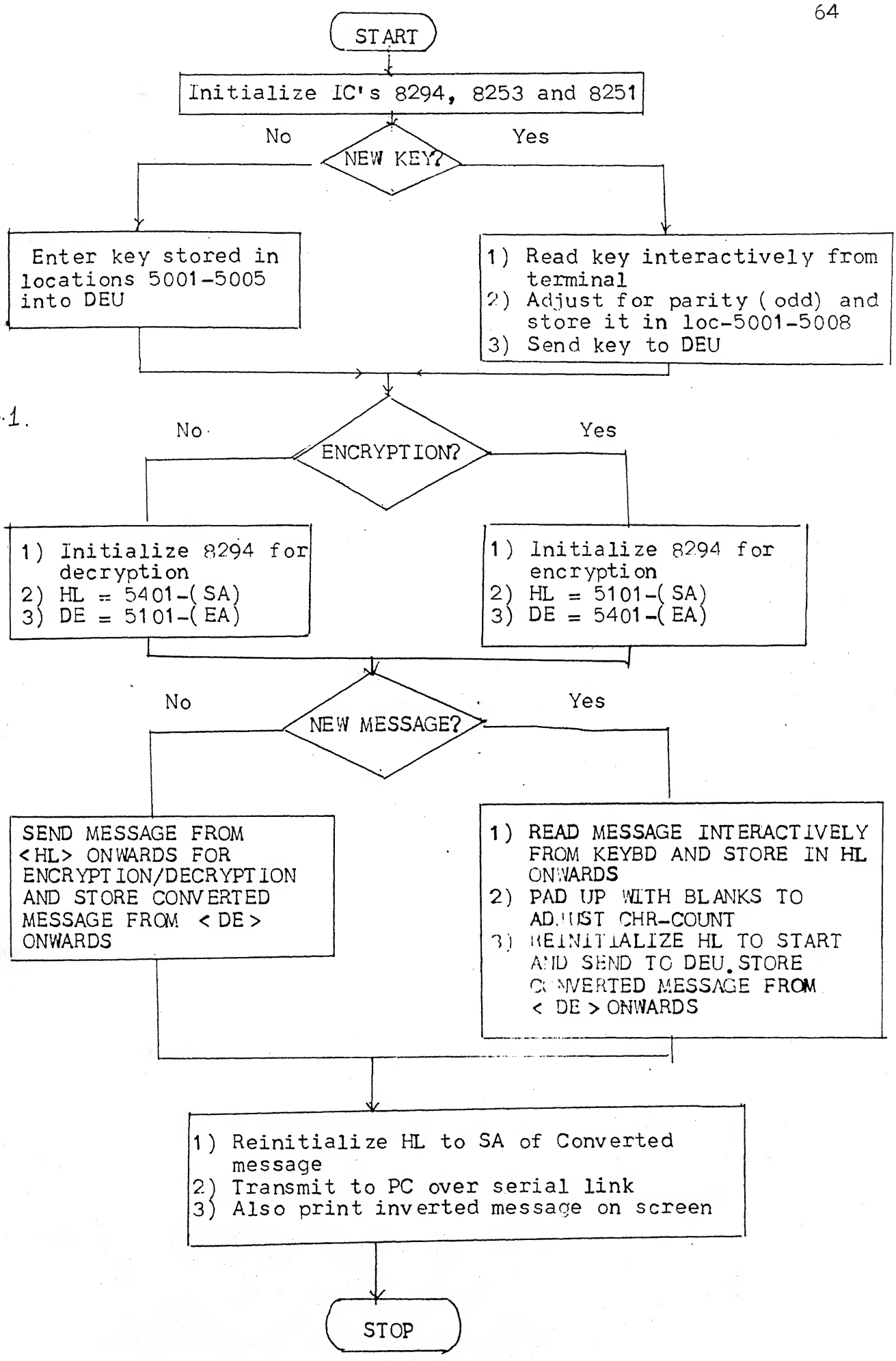

START
Initialize IC's 8294, 8253 and 8251
No
NEW KEY?
Yes
Enter key stored in locations 5001-5005 into DEU
1) Read key interactively from terminal
2) Adjust for parity (odd) and store it in loc-5001-5008
3) Send key to DEU
No
ENCRYPTION?
Yes
1) Initialize 8294 for decryption
2) HL = 5401-(SA)
3) DE = 5101-(EA)
1) Initialize 8294 for encryption
2) HL = 5101-(SA)
3) DE = 5401-(EA)
No
NEW MESSAGE?
Yes
SEND MESSAGE FROM <HL> ONWARDS FOR ENCRYPTION/DECRYPTION AND STORE CONVERTED MESSAGE FROM <DE> ONWARDS
1) READ MESSAGE INTERACTIVELY FROM KEYBD AND STORE IN HL ONWARDS
2) PAD UP WITH BLANKS TO ADJUST CHR-COUNT
3) REINITIALIZE HL TO START AND SEND TO DEU. STORE CONVERTED MESSAGE FROM <DE> ONWARDS
1) Reinitialize HL to SA of Converted message
2) Transmit to PC over serial link
3) Also print inverted message on screen
STOP

fig. 4.3.1.

CHAPTER 5

SOFTWARE IMPLEMENTATION OF DES

Despite the many advantages with hardware encryption schemes (especially in respect of speed of encryption) it sometimes becomes necessary to implement encryption in software, especially in cases where encryption is desired to be incorporated at a higher level in the hierarchy of network protocols, that is, when an end-to-end encryption scheme is preferred to data link encryption. With end to end encryption the user can change the encryption method whenever he feels that the old one has been compromised and naturally it is easier to do so if the encryption has been implemented in software.

In this chapter, an implementation of DES in software for the IBM personal computer has been described. The programming language used is TURBO PASCAL (an enhanced version of Standard PASCAL).

The program implements DES in the Electronic Code Book Mode, that is, it treats the message to be encrypted as a concatenation of 64 bit blocks and encrypts the blocks separately. There is no linking up of the blocks as in the cipher block chaining or cipher feedback modes of operation (described in detail is Chapter 3). This mode has been chosen mainly with ease of implementation in mind; in any

realistic implementation of a cipher system this might prove to be inadequate for reasons mentioned in Chapter 3.

The core of any of the more realistic modes, however, is the ECB mode, and it can be therefore easily improved upon to meet the security risks of a practical system.

As described in Chapter 3, the DES algorithm operates on a 64 bit input and enforces substitutions and transpositions on it under a 56 bit-key to arrive at a 64 bit output. There are a few basic operations which the algorithm repeatedly uses. Subroutines have been written to perform these basic operations. A flow chart based description of the various procedures and the entire algorithm follows.

5.1 Inputting the Substitution and Permutation Tables : The 'S-read' and 'p-read' routines

The permutation tables and substitution boxes of the DES algorithm are fixed and are therefore permanently stored in a file called INP-1, the tables being entered in the order in which they are 'read'.

The routines 'p-read' and 'S-read' perform the reading operation of the permutation tables, and the substitution boxes respectively from the file INP-1 into the program.

The 'p-read' routine reads into a one dimensional array of appropriate length and 'name' (which are transferred to

the routine as parameters) entries from the input file.

The S-read routine reads into a two dimensional array of standard dimension of 4x16 (all substitution boxes are of these dimensions, only the 'name' is passed as a parameter to the procedure) entries (numbers) from the input file.

5.2 The Procedure 'Permute'

The algorithm effects permutations on the input and the key and at various stages in the encryption/decryption process. The permutations are all fixed and are specified by the permutation tables, which are read into the program into one dimensional arrays by the "p-read" routine described above. The 'permute' routine, when invoked, effects the appropriate permutation.

The procedure has as its input parameters:

(i) the vector on which permutation is to be effected

(ii) the permutation table (which is also in the form of a one dimensional array)

(iii) the length of the array.

The permuted output can be in the same array as the input or is a different array, and is also specified as a variable output parameter.

5.3 Procedure Hex

It is most convenient to enter the key in Hexadecimal. The 'Hex' subroutine returns the decimal equivalent of a Hex character (which is eventually converted into its binary equivalent and stored in the key-array).

5.4 Procedure L-Shift

A look at Fig. 3.1.3 shows that each of the 16 basic iterations of the DES algorithm uses a different key derived from the original key through a series of left shifts according to the key schedule specified in table 3.1.8. When the same algorithm is used for decryption, the keys are furnished in the reverse order, i.e., the first iteration uses what would have been K_{16} for encryption, and so on. It is not necessary however, to calculate all the sixteen iteration of the key and store them; the 'decryption keys' can be obtained directly by suitably right shifting the original key. For instance K_{16} is arrived at after 28 left circular left shifts on the left and the right halves of the 56 bit key separately. K_{16} is therefore the original key. K_{15} is obtained after 27 left circular shifts. K_{15} is therefore obtained while decryption (it is K_2 in this case) by right circular shifting K once.

More generally, if K_i is arrived at by N left shifts on K_i, it can also be obtained by (28-N) right shifts on K.

(The shifts are all circular), or vice versa. We can therefore calculate a different key schedule based on right shifts which will be used while decryption. The procedure L-shift can be used to Left circular shift-or right circular shift a key register. Only the parameter which gives the number by which a register has to be shifted has to appropriately specified.

5.5 Procedure Iterate (n1, Li, ri, Key C, key d, encryp, p,e, pc-2, S1, S2,....,S8

This procedure is the core of the program. It computes the basic iteration of the DES algorithm (please refer fig. 3.1.1 for the main flow chart). In essence, the 'iterate' routine basically computes $f(r_i, k_i)$ (r_i, of the previous iteration, and K_i of the present iteration) and adds it modulo 2 to l_i of the previous iteration (l_i, and r_i are the left and the right halves of the data of the previous iteration). R_i remains unchanged. The parameter description is as follows:

n1: This specified the number of shifts on the key of the previous iteration to arrive at the key of the present iteration.

li, ri: The left and the right halves of the data vector of the previous iteration; both, 32 wide one dimensional arrays.

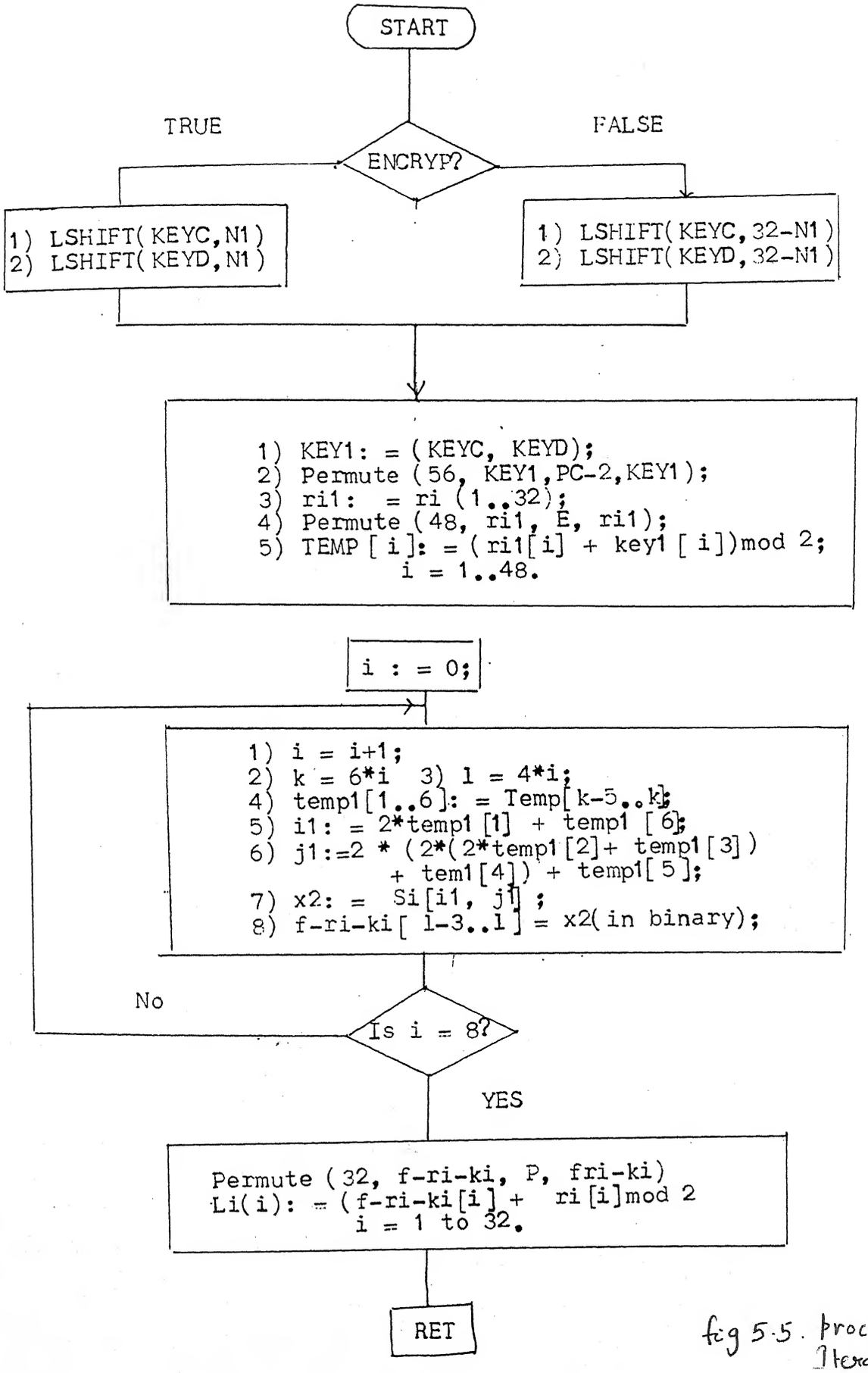


fig 5.5. Procedure Iterate.

key C, key d: The left and the right halves respectively of the key of the previous iteration; both 28-wide one dimensional arrays.

Encryp: The boolean variable which indicates how to compute the key for the present iteration, i.e., by left shifting or right shifting, according as encryp is true (encryption) or false (decryption).

p,e,pc-2: are the permutation vectors used by the routine.

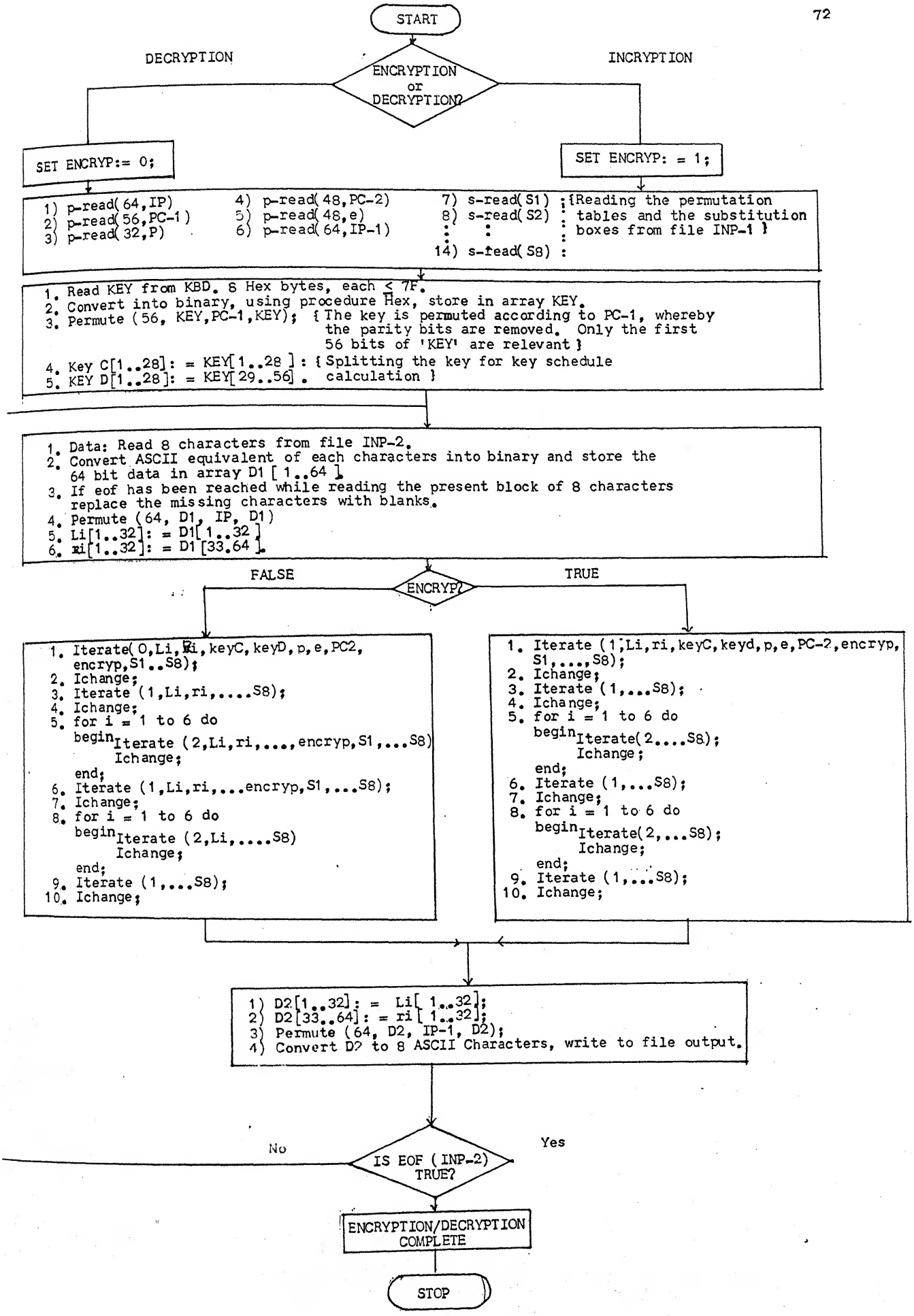
S1,...,S8: are the substitution boxes.

5.6 Procedure i Change

This procedure interchangesthe left and the right valves of the data output of the present iteration. The parameters are l_i, r_i, 32 bit registers; they are returned to the program after their contents are interchanged.

5.7 The Flow Diagram:

The program requires the message to be encrypted, to be entered in an input file named 'inp-2'. Each character in the file is expanded into its 8 bit ASIII equivalent and 8 characters are concatenated to form a 64 bit block.


START
DECRYPTION
ENCRYPTION or DECRYPTION?
INCRYPTION
SET ENCRYP:= 0;
SET ENCRYP: = 1;
1) p-read(64,IP)
2) p-read(56,PC-1)
3) p-read(32,P)
4) p-read(48,PC-2)
5) p-read(48,e)
6) p-read(64,IP-1)
7) s-read(S1) ;{Reading the permutation
8) s-read(S2) : tables and the substitution
14) s-read(S8) : boxes from file INP-1 }
1. Read KEY from KBD. 8 Hex bytes, each < 7F.
2. Convert into binary, using procedure Hex, store in array KEY.
3. Permute (56, KEY,PC-1,KEY); { The key is permuted according to PC-1, whereby the parity bits are removed. Only the first 56 bits of 'KEY' are relevant }
4. Key C[1..28]: = KEY[1..28] : { Splitting the key for key schedule
5. KEY D[1..28]: = KEY[29..56] . calculation }
1. Data: Read 8 characters from file INP-2.
2. Convert ASCII equivalent of each characters into binary and store the 64 bit data in array D1 [1..64]
3. If eof has been reached while reading the present block of 8 characters replace the missing characters with blanks.
4. Permute (64, D1, IP, D1)
5. Li[1..32]: = D1[1..32]
6. ri[1..32]: = D1[33.64]
FALSE
ENCRYP?
TRUE
1. Iterate(0,Li,ri, keyC, keyD, p, e, PC2, encryp,S1..S8);
2. Ichange;
3. Iterate (1,Li,ri,....S8);
4. Ichange;
5. for i = 1 to 6 do
begin Iterate (2,Li,ri,...,encryp,S1,...S8)
Ichange;
end;
6. Iterate (1,Li,ri,...encryp,S1,...S8);
7. Ichange;
8. for i = 1 to 6 do
begin Iterate (2,Li,....S8)
Ichange;
end;
9. Iterate (1,...S8);
10. Ichange;
1. Iterate (1,Li,ri,keyC,keyd,p,e,PC-2,encryp, S1,...,S8);
2. Ichange;
3. Iterate (1,...S8);
4. Ichange;
5. for i = 1 to 6 do
begin Iterate(2....S8);
Ichange;
end;
6. Iterate (1,...S8);
7. Ichange;
8. for i = 1 to 6 do
begin Iterate(2,...S8);
Ichange;
end;
9. Iterate (1,...S8);
10. Ichange;
1) D2[1..32]: = Li[1..32];
2) D2[33..64]: = ri[1..32];
3) Permute (64, D2, IP-1, D2);
4) Convert D2 to 8 ASCII Characters, write to file output.
No
IS EOF (INP-2) TRUE?
Yes
ENCRYPTION/DECRYPTION COMPLETE
STOP

The key is entered in Hexadecimal interactively on being prompted on the terminal. The flow chart, p.72. which is self explanatory, describes the other salient features of the program.

The output of the encryption/decryption process is in file named OUTPUT.

5.8 Software for Handling Communication

Encrypted data is transferred to and from the PC via the communications adapter provided. Programs for handling communications with the workstation have been written in BASIC in view of the fact that one can directly open the communication buffer as a text file and issue 'input' or 'print' commands for receiving or transmitting respectively. The "OPEN COM" instruction helps the user to specify the particulars of the asynchronous communication viz., the baud rate, number of data bits, number of stop bits, nature of parity, etc. Communication is through the 25 pin D shell connector provided at the back of the PC [15].

User Instruction:

The "TURBO-87" command to the PC in the 'DOS' mode enables the user to run his PASCAL programs. The data to be encrypted or decrypted is entered into a file named "INP2" (using the E command in the TURBO mode). Program DES.PAS acts upon the message in inp2,8 characters at a time,

filling up the last block with blanks if necessary to make the blocks of encryption complete. The output of the encryption/decryption process is in file "OPUT" which can be seen by using the 'E' command. For reconverting the data in 'OPUT' into original form, one will have to transfer it into file "INP2" and thereafter run "DES PAS" . The transfer can be effected by running "CHAN.PAS" once. File "INP1" has permanently the tables necessary for substitutions and transpositions. The user must ensure that the file "INP1" is also on the hard disk/discrete before running DES.PAS.

For communication with the work station, however, one has to 'enter' BASIC from DOS and use COM1. BAS for transmitting Data and COM2.BAS for receiving Data. The former transmits data from 'OPUT' (output of the encryption process) over the serial link, and COM2.BAS receives data over the serial link and stores it in "INP2" whereafter it is ready for encryption/decryption.

CHAPTER 6

CONCLUSIONS

The main objective of the present thesis has been to describe the implementations of DES both in hardware (using the Intel 8294-A Data encryption unit) and software (implementing the algorithm directly in TURBO PASCAL for the IBM PC).

Both the implementations have been found to be working satisfactorily. Messages can be encrypted in hardware from the workstation and transmitted over the serial communication link to the PC using the programs listed in Appendix-1. The message can be operated upon by the reverse transformation in the PC (the programs are listed in Appendix-2, with user instructions in Chapter 5), thereby verifying the correctness of the encryption software. The same could be done in the other direction also, viz. messages encrypted using the software in PC and thereafter transmitted to the workstation, to be processed by the hardware interface card.

In both the schemes, encryption was effected on messages in the Electronic Code Book fushion, i.e. treating the message as a concatenation of 64 bit blocks and encrypting the blocks separately.

The software for handling the communication between the PC and the workstation is in BASIC because of the provision there exists for reading from the communication adapter directly (or writing into it) after opening it as a text file. The program initiates the adapter for a data rate of 1200 baud and 8 bits/character and 2 stop bits asynchronous communication. The 8251 chip on the workstation card is programmed accordingly for 1200 baud (X 16 operation) 8 data bits and 2 stop bits.

6.1 Scope for Further Work

DES could be implemented in some of the more sophisticated modes of operation, such as the CBC (cipher block chaining) or the CFB (cipher feedback) mode. The ECB method reported here could be considered insecure for certain purposes for reasons described in Chapter 3.

The 8294-A, however, does not offer the facilities of CFB or CBC directly. There are more expensive chips (listed in Chapter 3) which offer this facility. But one could still implement these with provision for additional storage like shift registers on the card.

To modify the PASCAL program for CBC or CFB is a simple matter and should pose no difficulties.

Secondly, the software implementation could be more effective if the assembly language of the processor concerned is used for programming instead of a high level language like PASCAL. Although it would not match the hardware scheme in

speed of encryption, it would still improve upon the present speed substantially.

Use of a standard like the DES has advantages as well as pitfalls. The algorithm is certainly intricate enough to defeat even elaborate attempts at cryptanalysis but in view of the fact that the design details are not publicly announced one could suspect the presence of trapdoors in the algorithm.

For applications where a relatively unstudied standard like DES may be considered insecure, users must have additional or alternative encryption mechanisms.

REFERENCES

1. Denning, D.E.R., "Cryptography and Data Security", Addison Wesley, 1982.

2. Shannon, C.E., "Communication Theory of Secrecy Systems", Bell. Syst. Tech.J., Vol. 28, pp. 656-715 (Oct.'49).

3. Diffie, W. and Hellman, M., "New Directions in Cryptography", IEEE Trans. on Info. Theory, Vol. IT-22(6), pp. 644-654 (Nov. 1976).

4. Davies, D.W. and Price, W.L., "Security for Computer Networks", John Wiley and Sons, 1984.

5. Sethuraman, M., "Cipher Systems Based on Cyclic Difference Sets", EE. M.Tech. Thesis, Feb. '1986, IIT Kanpur.

6. Rivest, R.L., Shamir, A. and Adleman, L., "A Method for Obtaining Digital Signatures and Public Key Cryptosystems", Comm. ACM, Vol. 21(2), pp. 120-126, (Feb. 1978).

7. Campbell, C.M., "Design and Specification of Cryptographic Capabilities", IEEE Comm. Soc. Mag., Nov. 1978, Vol. 16, No. 6, pp. 15-19.

8. Tanenbaum, A.S., "Computer Networks", Prentice Hall, 1981.

9. Konheim, A.G., "Cryptography : A Primer", John Wiley & Sons, NY (1981).

10. Hellman, M.E., "DES will be Totally Insecure within Ten Years", IEEE Spectrum, Vol.16, pp. 32-39 (July, 1979).

11. Davies, D.W., Davida, G.I., "Hellman's Scheme Breaks DES in its Basic Form", IEEE Spectrum, Vol. 16, p. 39.

12. Turbo PASCAL, User Manual - Microsoft Co. Ltd.

13. PASCAL User Manual and Report - Jensen and N. Wirth.

14. Handbook of Microcomputer Peripherals,Intel Corporation, USA.

15. The IBM Personal Computer Technical Reference Manual, IBM Corpn, NY.

A P P E N D I X 1
THE PROGRAM IMPLEMENTING THE DES ALGORITHM

```
program DES;
{this program encrypts text message using the DES algorithm .the input
message to the encryption/decryption process must be entered in text
file inp2;the output is printed in text file oput.For its operation, the
program also needs the file inp1 to be present in the directory;this
file contains the permutation tables and the substitution boxes
necessary for the algorithm.}

label 10,15,16,17,20,25,100;
type
     array1=array[1..64] of integer;
     array2=array[1..56] of integer;
     array3=array[1..48] of integer;
     array4=array[1..32] of integer;
     array5=array[1..28] of integer;
     array6=array[1..16] of integer;
     array7=array[1..16] of char;
     array8=array[0..3,0..15] of integer;
     array9=array[1..8] of integer;
     ring=string[4];
     num=integer;
var
     ip,d1,ip1,d2,d3,key:array1;
     e,pc1,pc2:array1;
     li,li1,ri2,ri,p:array1;
     keyc,keyd:array5;
     mes,keyn,mes1,x6:array7;
     kls:array6;
     X3,x4:char;
     i,i1,j,j1,j2,k,k1,x1,x2,x5,p1,rem:integer;
     f,g,h:text;
     inp1,inp2,oput,tex:ring;
     encryp,flag:boolean;
     s1,s2,s3,s4,s5,s6,s7,s8:array8;

procedure hex(x4:char;var x5:integer);{this procedure converts the hex
characters used in the inputting the key to their corresponding decimal
values}
var x6:integer;
begin
      case x4 of
      'A':x5:=10;
      'B':x5:=11;
      'C':x5:=12;
      'D':x5:=13;
      'E':x5:=14;
      'F':x5:=15;
      '0','1','2','3','4','5','6','7','8','9':val(x4,x5,x6)
      end;
end;
procedure pread(ni,n:integer;var a:array1);{procedure to read the
```

```
permutation tables in file inp1. into one dimensional arrays into the
program.}
begin
      for i := 1 to n do
      begin
            read(f,a[i]);{write(a[i]:3);
            if i mod n1=0 then writeln;}
      end;
end;
procedure sread(var a:array8);{procedure to read the substitution boxes
into two-dimensional arrays into the program}
var a1:array8;
begin
      for i:=0 to 3 do
      begin
      for j:=0 to 15 do
      begin
            read(f,a1[i,j]);a[i,j]:=a1[i,j];
            {write(a[i,j]:3);}
      end;
      {writeln;}
      end;
end;
procedure permute(n1,n2:integer;d1,p:array1;var d2:array1);{procedure to
permute a given vector(d1) according to a specified permutation
table(p);the output of the permutation is in vector d2}
var d3:array1;i,j:integer;
begin
      for i:=1 to n2 do
      begin
            j:=p[i];
            d3[i]:=d1[j];
      end;
      for i:=1 to n2 do
      d2[i]:=d3[i];
end;
procedure lshift(n:integer;var a:array5);{procedure to left shift/right
shift a given vector by a given number of places}
var l:array5;i,j:integer;
begin
      for i:= 1 to 28 do
      begin
            j:=i-n;
            if j<=0 then j:=i-n+28;
            l[j]:=a[i];
      end;
      for i:=1 to 28 do
      a[i]:=l[i];
end;
procedure ichange(var ri2,li:array1);{procedure to interchange the left
and right halves of the data vector after computation of the function
"f(Ri-1,Ki)"}
var lii:array1;
begin
      for i:=1 to 32 do
```

```
      begin
           li1[i]:=li[i];
           li[i]:=ri2[i];
           ri2[i]:=li1[i];
      end;
end;

procedure iterate(n1:integer;var keyc,keyd:array5;encryp:boolean;
    var ri2,li:array1;s1,s2,s3,s4,s5,s6,s7,s8:array8;pc2,e,p:array1);
var li1,ri,friki,key1:array1;temp:array3;temp1:array9;
ts:array8;K1,i,i1,j1,j,k,l,x1:integer;
{this procedure constitutes the heart ofthe program;it computes the
function f(Ri-1,Ki) returns the data vectors to the program.It takes all
those permutation tables and s-boxes as input parameters and returns
the output data vectors to the program.}
begin
      if encryp then
      begin
           {writeln('encryption');}
           lshift(n1,keyc);
           lshift(n1,keyd);
      end  else
      begin
           {writeln('decryption');}
           n1:=28-n1;
           lshift(n1,keyc);
           lshift(n1,keyd);
      end;
      {write('keyc:   ');
      for i:=1 to 28 do
      write(keyc[i]:1);writeln;
      write('keyd:   ');
      for i:=1 to 28 do
      write(keyd[i]:1);writeln;}
      for i:=1 to 28 do
      begin
           key1[i]:=keyc[i];
           key1[i+28]:=keyd[i];
      end;
      permute(56,48,key1,pc2,key1);
      {write('key1:   ');
      for i:=1 to 48 do write(key1[i]:1);writeln;}
      i:=0;
      {repeat
      i:=i+1;k1:=8*i;
      x1:=key1[k1-7];
      for j:=k1-7 to k1-1 do
      x1:=2*x1+key1[j+1];
      write(x1:6);
      until i=6;}
      for i:=1 to 32 do
      ri[i]:=ri2[i];
      {write('ri:   ');
      for i:=1 to 32 do write(ri[i]:1);writeln;}
      permute(32,48,ri,e,ri);
```

```
{for i:=1 to 48 do write(ri[i]:1);write('    :ri permuted');writeln;}
for i:=1 to 48 do
temp[i]:=(ri[i]+key1[i]) mod 2;
{for i:=1 to 48 do write(temp[i]:1);writeln;}
i:=0;
repeat
        i:=i+1;
        j:=6*i;
        l:=4*i;
        for k:=6 downto 1 do
        begin
             temp1[k]:=temp[j];
             j:=j-1;
        end;
        for i1:=0 to 3 do
        for j1:=0 to 15 do
        case i of
             1:ts[i1,j1]:=s1[i1,j1];
             2:ts[i1,j1]:=s2[i1,j1];
             3:ts[i1,j1]:=s3[i1,j1];
             4:ts[i1,j1]:=s4[i1,j1];
             5:ts[i1,j1]:=s5[i1,j1];
             6:ts[i1,j1]:=s6[i1,j1];
             7:ts[i1,j1]:=s7[i1,j1];
             8:ts[i1,j1]:=s8[i1,j1]
        end;
        {for i1:=0 to 3 do
        begin
        for j1:=0 to 15 do
        write(ts[i1,j1]:3);writeln;
        end;}
        i1:=temp1[6]+2*temp1[1];
        j1:=2*(2*(2*temp1[2]+temp1[3])+temp1[4])+temp1[5];
        x2:=ts[i1,j1];
        for k:=1 to 4 do
        begin
             friki[l]:=x2 mod 2;
             x2:=x2 div 2;
             l:=l-1;
        end;
until i=8;
permute(32,32,friki,p,friki);
for i:=1 to 32 do
li[i]:=(friki[i]+li[i]) mod 2;
end;
{-------------------------------------------------------------------------------
        MAIN PROGRAM BEGINS HERE
-------------------------------------------------------------------------------}
begin
     assign(f,'inp1');
     assign(g,'inp2');
     assign(h,'oput');
     reset(f);
     reset(g);
     rewrite(h);
```

```
writeln('PLEASE TYPE "E" FOR ENCRYPTION OR ANY OTHER KEY FOR
DECRYPTION.');
writeln;writeln;
read(KBD,x3);
if  (x3='E')or (x3='e') then
begin
      encryp:=true;
      writeln('e  n  c  r  y  p  t  i  o  n  ');
end else
begin
       writeln('d  e  c  r  y  p  t  i  o n' );
       encryp:=false;
end;
writeln;writeln;
writeln('PLEASE ENTER THE KEY:8 BYTES IN HEX, EACH <=7F.M-S-BYTE
FIRST.');
writeln;

for i:=1 to 64 do
key[i]:=0;
i:=0;i1:=0;
repeat                  {reading the key interactively from the terminal}
       i:=i+1;
       j:=8*i;
       i1:=i1+1;
       read(KBD,X4);
       hex(X4,x5);write(x5:2);
       j:=j-5;
       for k:=1 to 3 do
       begin
            key[j]:=x5 mod 2;
            x5:=x5 div 2;
            j:=j-1;
       end;
       j:=j+7;
       i1:=i1+1;
       read(KBD,X4);
       hex(X4,x5);write(x5:2,'     ');
       for k:=1 to 4 do
       begin
            key[j]:=x5 mod 2;
            x5:=x5 div 2;
            j:=j-1;
       end;
       j:=i*8;
       for k1:=j-7 to j do write(key[k1]:1);
       writeln;
until  i=8;writeln;
{for i:=1 to 64 do
write(key[i]:1);
writeln;}
clrscr;
if encryp then writeln('Encryption being conducted.PLEASE WAIT.')
else
    writeln('Decryption being conducted.PLEASE WAIT.');
```

```
pread(7,56,pc1);
permute(64,56,key,pc1,key);
{for i:=1 to 56 do
write(key[i]:1);writeln;}
pread(8,64,ip);                    {inputting the tables}
pread(6,48,e);
pread(6,48,pc2);
pread(4,32,p);
sread(s1);
sread(s2);
sread(s3);
sread(s4);
sread(s5);
sread(s6);
sread(s7);
sread(s8);
pread(8,64,ip1);
flag:=false;rem:=0;
repeat{reading one block of 8 characters from the text file inp2.
TURBO PASCAL treats ASCII(26) as an EOF character;so if any of the
characters in the output of the encryption process happens to be
ASCII(26) the text output file will be abruptly closed and
subsequent characters lost.Therefore whenever the output has
ASCII(26) as one constituent character a string "*" is printed in
the output file.Correspondingly,if the input file has "*" in it,it
must be replaced with char(26) when sending for encryption.This
part takes care of such eventualities.The eventuality where the
string "*" staggers the boundary between two adjacent blocks is
also met.}
      15:for i:=1 to 28 do
      begin
           keyc[i]:=key[i];
           keyd[i]:=key[i+28];
      end;
      100:I:=0;I1:=0;
      repeat
           if rem=1 then
           begin
           p1:=1;x6[1]:=x6[2];rem:=0;goto 17;
           end;
           if rem=2 then
           begin
           p1:=2;x6[1]:=x6[2];x6[2]:=x6[3];rem:=0;goto 17;
           end;
           if rem=3 then
           begin
           p1:=1;x6[1]:=x6[3];rem:=0;goto 17;
           end;
           if eof(g) then
           begin
                flag:=true;
                goto 10;
           end;
           read(g,x4);x6[1]:=x4;
           if x4='*' then
```

```
        begin
        read(g,x4);x6[2]:=x4;
        if x4='*' then
        begin
        read(g,x4);x6[3]:=x4;
        if x4='"' then
        begin
        x6[1]:=char(26);p1:=1;
        end else p1:=3;
        end else p1:=2;
        end else p1:=1;
        rem:=0;
        if (p1=2) and (i=7) then
        begin
        p1:=1;
        rem:=1;
        end;
        if (p1=3) and (i=7) then
        begin
        p1:=1;
        rem:=2;
        end;
        if (p1=3) and (i=6) then
        begin
        p1:=2;
        rem:=3;
        end;
        17:for j2:= 1 to p1 do
        begin
        i:=i+1;
        k1:=8*i;
        x1:=ord(x6[j2]);{rite(x6[j2]);writeln(x1);}
        for j1:=k1 downto (k1-7) do
        begin
            d1[j1]:=x1 mod 2;
            x1:=x1 div 2;
        end;
        end;
until i=8;if eof(g) then flag:=true;goto 16;
     10:for i:=i+1 to 8 do
     begin
     k1:=8*i;
     x1:=32;{write(char(x1));writeln(X1);}
     for j1:=k1 downto (k1-7) do
     begin
          d1[j1]:=x1 mod 2;
          x1:=x1 div 2;
     end;
end;

{for i:=1 to 64 do write(d1[i]:1);
writeln;}
{for i:=1 to 28 do write(keyc[i]:1);writeln;
for i:=1 to 28 do write(keyd[i]:1);writeln;}
16:permute(64,64,d1,ip,d2);
```

```
{for i:=1 to 64 do write(d2[i]:1);writeln;}
for j1:=1 to 32 do
begin
     li[j1]:=d2[j1];
     ri2[j1]:=d2[j1+32];
end;
{for i:=1 to 32 do write(li[i]:1);writeln;
for i:=1 to 32 do write(ri[i]:1);writeln;}
if encryp then
begin {calling "iterates according to key schedule}
     {writeln('encryption');}
     for j1:=1 to 2 do
     begin
          iterate(1,keyc,keyd,encryp,ri2,li,s1,s2,s3,s4,s5,s6,
          s7,s8,pc2,e,p);
          ichange(li,ri2);
     end;
     for j:=1 to 6 do
     begin
          iterate(2,keyc,keyd,encryp,ri2,li,s1,s2,s3,s4,s5,s6
          ,s7,s8,pc2,e,p);
          ichange(li,ri2);
     end;
     iterate(1,keyc,keyd,encryp,ri2,li,s1,s2,s3,s4,s5,s6,s7,s
     8,pc2,e,p);
     ichange(li,ri2);
     for i:=1 to 6 do
     begin
          iterate(2,keyc,keyd,encryp,ri2,li,s1,s2,s3,s4,s5,s6
          ,s7,s8,pc2,e,p);
          ichange(li,ri2);
     end;
     iterate(1,keyc,keyd,encryp,ri2,li,s1,s2,s3,s4,s5,s6,s7,s
     8,pc2,e,p);
end else
begin
     {writeln('decryption');}
     iterate(0,keyc,keyd,encryp,ri2,li,s1,s2,s3,s4,s5,s6,s7,s8,
     pc2,e,p);
     ichange(li,ri2);
     iterate(1,keyc,keyd,encryp,ri2,li,s1,s2,s3,s4,s5,s6,s7,s8,
     pc2,e,p);
     ichange(li,ri2);
     for i:=1 to 6 do
     begin
          iterate(2,keyc,keyd,encryp,ri2,li,s1,s2,s3,s4,s5,s6,
          s7,s8,pc2,e,p);
          ichange(li,ri2);
     end;
     iterate(1,keyc,keyd,encryp,ri2,li,s1,s2,s3,s4,s5,s6,s7,s8,
     pc2,e,p);
     ichange(li,ri2);
     for i:=1 to 6 do
     begin
          iterate(2,keyc,keyd,encryp,ri2,li,s1,s2,s3,s4,s5,s6,
```

```
                    s7,s8,pc2,e,p);
                    ichange(li,ri2);
               end;
               iterate(i,keyc,keyd,encryp,ri2,li,s1,s2,s3,s4,s5,s6,s7,s8,
               pc2,e,p);
          end;
          {for j:=1 to 32 do
          begin
          li2[j]:=ri[j];
          ri2[j]:=li[j];
          end;}
          for j:=1 to 32 do
          begin
               d2[j]:=li[j];
               d2[32+j]:=ri2[j];
          end;
          permute(64,64,d2,ip1,d1);
          {for i:=1 to 64 do write(d1[i]:1);writeln;
          for i:=1 to 64 do write(d2[i]:1);writeln;}
          i:=0;
          repeat{converting the output to ASCII for writing to output
          file}
               i:=i+1;
               k1:=8*i;
               x1:=d1[k1-7];
               for j:=(k1-7) to (k1-1) do
               x1:=2*x1+d1[j+1];{write(x1:6);writeln;}
               {if(x1<32) or (x1>126) then
               x1:=63;}
               if x1=26 then write(h,'"*"') else
               write(h,char(x1));{write(char(x1));}
          until i=8;
          {for i:=1 to 64 do
          begin
               d3[i]:=d2[i];
               d2[i]:=d1[i];
               d1[i]:=d3[i];
          end;}
     until flag;
     clrscr;
     writeln('The input message is:');
     writeln;
     reset(g);
     while not eof(g)
     begin
          read(g,x4);
          write(x4);
     end;writeln;
     writeln('-------------------------------------------------------------------------------');
     writeln;
     if encryp then writeln('The encrypted message is:') else
     writeln('The decrypted message is:');
     writeln;
     25:reset(h);
     while not eof(h)
```

```
        begin
        read(h,x4);write(x4);
        end;
        close(f);close(g);close(h);
end.
```

A P P E N D I X 2

THE ASSEMBLY LANGUAGE PROGRAM FOR ENCRYPTION IN THE WORK-STATION USING THE HARDWARE ENCRYPTION INTERFACE.

```
4C00
JMP START
WAIT1:IN E7 ;routine invoked for waiting for DEU flag
ANI 02
JNZ WAIT1
RET
WAIT:PUSH B ;routine called when asking for new message or new key.
LXI B,MES1
CALL PNTMS
CALL CRLF
CALL READ
CPI 4E
POP B
RET
READS:CALL READ ;routine for reading message from terminal.
CALL PRINT
MOV M,A
CPI 7C
JZ OUT
INX H
JMP READS
OUT:DCX H
RET
MES1:DBA KEY/MESSAGE -- "N"EW  OR  "P"REVIOUSLY STORED ?*
MES2:DBA PLEASE ENTER THE KEY.*
MES3:DBA "E"NCRYPTION  OR  "D"ECRYPTION ?*
MES4:DBA PL. ENTER THE MESSAGE.END WITH *
MES5:DBA    BYTES*
RDATA:PUSH H ;routine for inputting new message,padding up with
PUSH B         requisite number of blanks to make number of characters
CALL READS     a multiple of 8.
LOOP4:MOV A,L
ANI 0F
JZ STEP8
CPI 08
JZ STEP8
INX H
MVI A,20
MOV M,A
JMP LOOP4
STEP8:MOV A,H
SUB B
MOV H,A
SHLD 53FE
CALL HLPNT
LXI B,MES5
CALL PNTMS
CALL CRLF
POP B
```

```
POP H
RET
SEND8:PUSH D ;routine for sending 8 bytes to DEU
MVI E,08
LOOP6:CALL WAIT1
MOV A,M
OUT E6
CALL BIHEX
CALL TWOSP
DCX B
INX H
DCR E
JNZ LOOP6
CALL CRLF
POP D
RET
READ8:PUSH B ;routine for reading 8 converted bytes from DEU
MVI E,08
LOOP7:IN E7
ANI 01
JZ LOOP7
IN E6
STAX D
CALL BIHEX
CALL TWOSP
INX D
DCR B
JNZ LOOP7
CALL CRLF
LOOP9:IN E7
ANI 08
JZ OUT1
IN E6
JMP LOOP9
OUT1:POP B
RET
PRTS:PUSH H ;routine for displaying the output message
LHLD 53FE    and transmitting over the serial link.
MOV B,H
MOV C,L
POP H
MVI D,F0
LOOPA:IN E5
ANI 02
JZ LOOPA
IN E4
CPI 11
JNZ LOOPA
XMIT:IN E5
ANI 01
JZ XMIT
MOV A,M
OUT E4
CALL PRINT
DCX B
```

```
INX H
DCR D
JNZ STEPC
MVI D,F0
JMP LOOPA
STEPC:XRA A
CMP B
JNZ XMIT
CMP C
JNZ XMIT
RET
START:CALL WAIT1
XRA A
OUT E7
OUT E5
OUT E5
OUT E5
MVI A,40
OUT E5
MVI A,6A
OUT E3
MVI A,BA
OUT E1
MVI A,CE
OUT E5
MVI A,37
OUT E5
CALL WAIT
JNZ STEP4
LXI B,MES2
CALL PNTMS
CALL CRLF
LXI H,5011
CALL HXCHR ;inputting the key
LXI H,5018  ;adjusting the key for odd parity
LXI D,5008
LOOP1:MOV A,M
MVI C,00
MVI B,08
LOOP2:RAR
JNC STEP1
INR C
STEP1:DCR B
JNZ LOOP2
MOV B,A
MOV A,C
ANI 01
JZ STEP2
MOV A,B
ANI FE
JMP STEP3
STEP2:MOV A,B
ORI 01
STEP3:STAX D
DCR E
```

```
DCR L
JNZ LOOP1
STEP4:CALL WAIT1
MVI A,40
OUT E7
LXI H,5000
LOOP3:CALL WAIT1
MOV A,M
OUT E6
DCR L
JNZ LOOP3
LXI B,MES3
CALL PNTMS
CALL CRLF
CALL READ
CPI 45
JNZ STEP5
CALL WAIT1
MVI A,30 ;initialising the DEU for encryption
OUT E7
LXI H,5101
LXI D,5401
MVI B,51
JMP STEP6
STEP5:CALL WAIT1
MVI A,20 ;initialising the DEU for decryption
OUT E7
LXI H,5401
LXI D,5101
MVI B,54
STEP6:CALL WAIT
JNZ STEP7
PUSH B
LXI B,MES4
CALL PNTMS
CALL CRLF
POP B
CALL RDATA
STEP7:PUSH D
PUSH H
LHLD 53FE
MOV B,H
MOV C,L
POP H
LOOP5:CALL SEND8
CALL READ8
XRA A
CMP B
JNZ LOOP5
CMP C
JNZ LOOP5
POP H
CALL PRTS
RST 1
[illegible]
```